MICROCOPY RESOLUTION TEST CHART
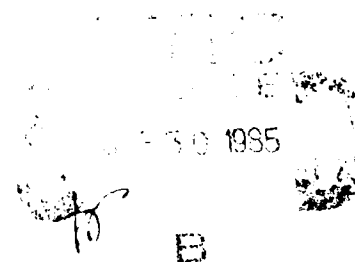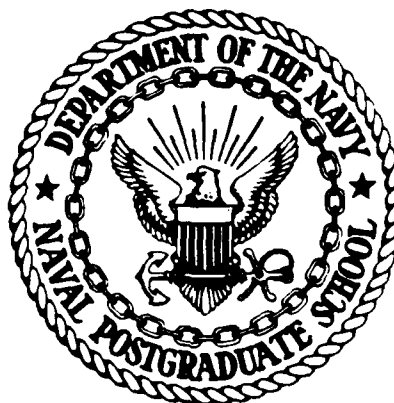NATIONAL BUREAU OF STANDARDS-1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

VLSI DESIGN OF A
SIXTEEN BIT PIPELINED MULTIPLIER
USING THREE MICRON NMOS TECHNOLOGY

by

Richard J. Simchik Jr.

June 1985

Thesis Advisor: H. H. Loomis

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. *A159 433*	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* VLSI Design of a Sixteen Bit Pipelined Multiplier Using Three Micron NMOS Technology		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1985
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard J. Simchik Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943-5100		12. REPORT DATE June 1985
		13. NUMBER OF PAGES 94
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

NMOS VLSI Design, Pipelined Multiplier, Two's Complement Multiplier, CAD Tools, MacPitts Silicon Computer

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The application of computer-aided design tools to the full custom design and testing of a 16-bit pipelined two's complement multiplier in three micron NMOS is described. A comparison between the full custom carry-save addition (CSA) multiplier designed using CAD tools and a multiplier generated by the MacPitts silicon compiler is presented. Additional background material is also presented on the CSA multiplication algorithm utilized.

DD FORM 1473   EDITION OF 1 NOV 65 IS OBSOLETE

1

**VLSI Design of a
Sixteen Bit Pipelined Multiplier
Using Three Micron NMOS Technology**

by

Richard J. Simchik Jr.
Captain, United States Army
B.S., Clarkson University, 1978

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 1985

Author: _____
Richard J. Simchik Jr.

Approved by: _____
Herschel H. Loomis, Thesis Advisor

_____
Donald E. Kirk, Second Reader

_____ for H. B. Rigas
Harriett B. Rigas, Chairman,
Department of Electrical and Computer Engineering

_____
John N. Dyer,
Dean of Science and Engineering

ABSTRACT

The application of computer-aided design (CAD) tools in the full custom design and testing of a 16-bit pipelined two's complement multiplier in three micron NMOS is described. A comparison between the full custom carry-save addition (CSA) multiplier designed using CAD tools and a multiplier generated by the MacPitts silicon compiler is presented. Additional background material is also presented on the CSA multiplication algorithm utilized. → used. Keywords:

TABLE OF CONTENTS

4

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

With the ever increasing demand for extremely complex
integrated circuits, today's electrical engineers and
systems designers have to be knowledgeable in the design and
fabrication of Very Large Scale Integrated (VLSI) circuits.
Several approaches exist today for the design of VLSI
circuits. These approaches include the interconnection of
standard library cells, gate arrays, programmable logic
arrays, and full custom design. Full custom design is the
most time consuming and expensive of the three, but gener-
ally yields a more efficient VLSI design in terms of circuit
density and speed of operation.

One methodology for full custom design that can be
easily understood and implemented by the systems designer
has been developed by Mead and Conway [Ref. 1]. This meth-
odology, coupled with the wide variety of computer-aided
design (CAD) tools that are available, makes it possible for
the systems designer to translate a design from a functional
block diagram, or a logic diagram, to silicon. Intelligent
simulation of the design prior to fabrication gives the
designer a high degree of confidence that the circuit func-
tions as desired, barring any unforeseen fabrication errors.

Another method that is available for the generation of
VLSI circuits is the use of a silicon compiler which takes
as input an algorithmic description of a circuit's desired
functions and generates the final layout of a VLSI circuit.
Using this approach to circuit design results in a rapid
design turn-around time. This allows the system designer
the ability to explore different architectures and find the
method best suited to solve a specific problem. One such
compiler that is installed and running at the Naval

8

Postgraduate School (NPS) is the MacPitts silicon compiler developed at Massachusetts Institute of Technology's Lincoln Laboratory. The installation and initial research on the MacPitts compiler is documented in work done previously by Carlson [Ref. 2]. Carlson utilized the MacPitts silicon compiler to generate an 8-bit unsigned pipelined multiplier to be used in a digital filter. To provide the basis for comparison of a full custom design and a design generated by the MacPitts silicon compiler, a 16-bit two's complement multiplier in three micron NMOS was hand-crafted using CAD tools currently available at NPS.

The discussion of a general carry-save addition (CSA) multiplier follows in Chapter 2. Chapter 3 presents the adaptation of the CSA multiplication scheme to the 16-bit two's complement multiplier. The remainder of Chapter 3 contains the design and testing of the multiplier and a description of the CAD tools utilized. Chapter 4 presents a test plan for the VLSI circuit after its fabrication by the MOS Implementation Service (MOSIS) of the Defense Advanced Research Projects Agency. This is followed by a comparison of the hand-crafted and MacPitts generated multipliers in Chapter 5.

II. UNSIGNED BINARY MULTIPLICATION

In this chapter, the implementation of an unsigned binary parallel multiplier is described. First, a brief discussion of the add-and-shift algorithm is presented. Although almost every reference in digital arithmetic contains a section on this algorithm (also called sequential multiplication), it is given here so that terminology and representations used in this chapter and the next may be introduced. Next, a multiplication scheme utilizing simultaneous generation of partial products followed by simultaneous reduction using carry-save addition (CSA) is described. The chapter concludes with a discussion of implementing this parallel multiplication scheme as a pipelined VLSI design.

A. ADD-AND-SHIFT ALGORITHM

The basis for the multiplier design presented in this chapter is the add-and-shift algorithm, which is similar to the way one multiplies using pencil and paper. For example, as shown in Figure 2.1, in multiplying two binary numbers each bit of the multiplier requires a corresponding add-and-shift operation.

A mathematical representation of the add-and-shift algorithm for two n-bit numbers is given in Equation 2.1. This equation has been derived from chapter 2 of _Introduction to Computer Architecture_ by Stone and others [Ref. 3].

$$P = \sum_{k=0}^{n-1} 2^k a_k b \qquad \text{(eqn 2.1)}$$

In this equation and throughout the remainder of this

10

```
MULTIPLICAND          1101
MULTIPLIER           x1011
                      1101
PARTIAL PRODUCTS      1101
                     0000
                    110 1
FINAL PRODUCT      10001111
```

Figure 2.1   Paper and Pencil Multiplication.

thesis, concatenation implies the logical AND, the symbol +
implies the logical OR, b represents the n-bit multiplicand
vector, a_n represents bit n of the multiplier vector a and P
represents the 2n bit product vector.   Figure   2.2 illus-
trates this concept for the multiplication of two 8-bit
operands and Figure 2.3 introduces a convenient dot repre-
sentation of the same multiplication.   As can be seen from
Figure 2.2, multiplying two 8-bit operands results in eight
partial products which are added to form a 16-bit final
product.

$$X_7X_6X_5X_4X_3\,X_2\,X_1X_0 \leftarrow \text{MULTIPLICAND}$$
$$Y_7Y_6Y_5Y_4Y_3\,Y_2\,Y_1Y_0 \leftarrow \text{MULTIPLIER}$$
$$A_7A_6A_5A_4A_3\,A_2\,A_1A_0 \leftarrow \text{A PARTIAL PRODUCT}$$
$$B_7B_6B_5B_4B_3B_2\,B_1\,B_0$$
$$C_7C_6C_5C_4C_3C_2C_1\,C_0$$
$$D_7\,D_6D_5D_4D_3D_2D_1D_0$$
$$E_7\,E_6\,E_5E_4E_3E_2E_1E_0$$
$$F_7\,F_6\,F_5\,F_4F_3F_2F_1F_0$$
$$G_7\,G_6\,G_5\,G_4\,G_3G_2G_1G_0$$
$$H_7\,H_6\,H_5\,H_4\,H_3\,H_2H_1H_0$$
$$S_{15}S_{14}S_{13}S_{12}S_{11}S_{10}S_9\,S_8\,S_7\,S_6\,S_5\,S_4\,S_3\,S_2\,S_1\,S_0 \leftarrow \text{FINAL PRODUCT}$$

Figure 2.2   Multiplying Two 8-bit Operands.
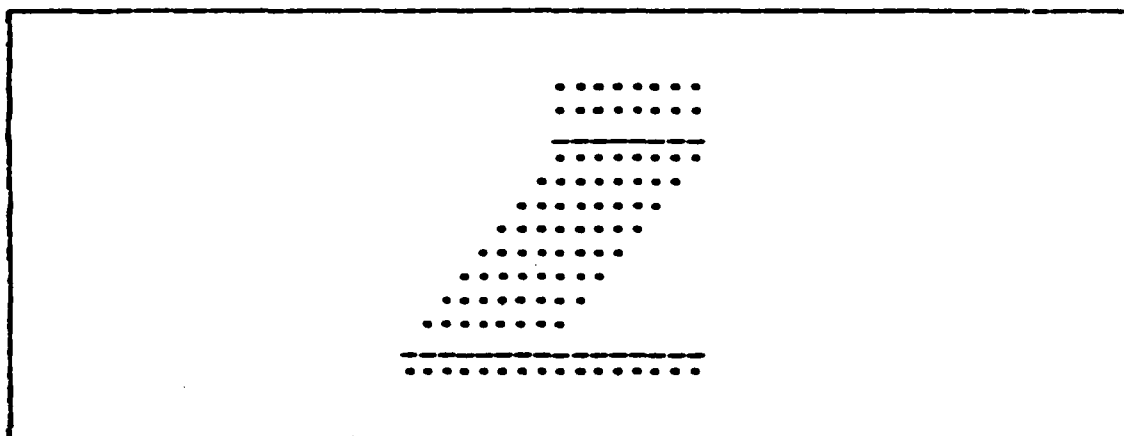
Figure 2.3   Dot Representation.

B. SIMULTANEOUS MATRIX GENERATION AND REDUCTION

In terms of speed, the basic add-and-shift algorithm is the slowest of the multiplication schemes. One method to improve the speed of the basic sequential multiplier is to perform as many operations as possible in parallel. This method, known as the Simultanoeus Matrix Generation and Reduction method [Ref. 4: pp. 132-147], is composed of three distinct steps. In the first step, all of the partial products are simultaneously generated. In the next step, the resultant matrix of partial products is reduced using carry-save addition (CSA) until two vectors remain. Finally, the two remaining vectors are added together to form the final product.

1. Partial Products Generation

The simplest way to generate each bit position of the partial products is to use the logical AND operation as a 1x1 multiplier. For example, in Figure 2.2, each of the terms in the eight partial products is the result of a logical AND operation and also corresponds to a single dot in each of the partial products of Figure 2.3. For an n-bit

12

multiplication this scheme requires nxn AND gates, which is a simple, but hardware intensive scheme.

It is possible to use encoding techniques that will reduce the number of partial products. One such method that reduces the number of partial products by half is the modified Booth's algorithm. For a description of both Booth's original and modified algorithms, the reader is referred to two presentations of these topics [Refs. 4,5: pp. 132-137, 152-157].

Another way to generate partial products is to use read only memories (ROMs). For example, the 8x8 multiplication of Figure 2.2 can be implemented using four 256x8 ROMs where each ROM performs a table lookup multiplication, as shown in Figure 2.4.

In Figure 2.4, the 4-bit value of each element of the pairs (Y0,X0), (Y0,X1), (Y1,X0), and (Y1,X1) is concatenated to form an 8-bit address into the ROM table. The ROM location corresponding to the address contains a unique 8-bit product. Thus four tables are required to simultaneously form the products Y1xX1, Y1xX0, Y0xX1, and Y0xX0. Note that the Y0xX0 and Y1xX1 terms have disjoint significance, thus only three terms must be added to form the final product. The number of rearranged partial products which must be summed is referred to as the matrix height h. This height corresponds to the number of initial inputs to the CSA tree. A generalization of this scheme for up to a 64x64 bit multiplication is shown in Figure 2.5. Each rectangle in Figure 2.5 [Ref. 4: p. 138] represents a 4x4 ROM multiplier product.

Table I [Ref. 4: p. 139] summarizes the maximum height of the partial products for the three partial product generation schemes discussed in this section.

In the final design implemented in this thesis, the partial products were generated using the 1x1 multiplier
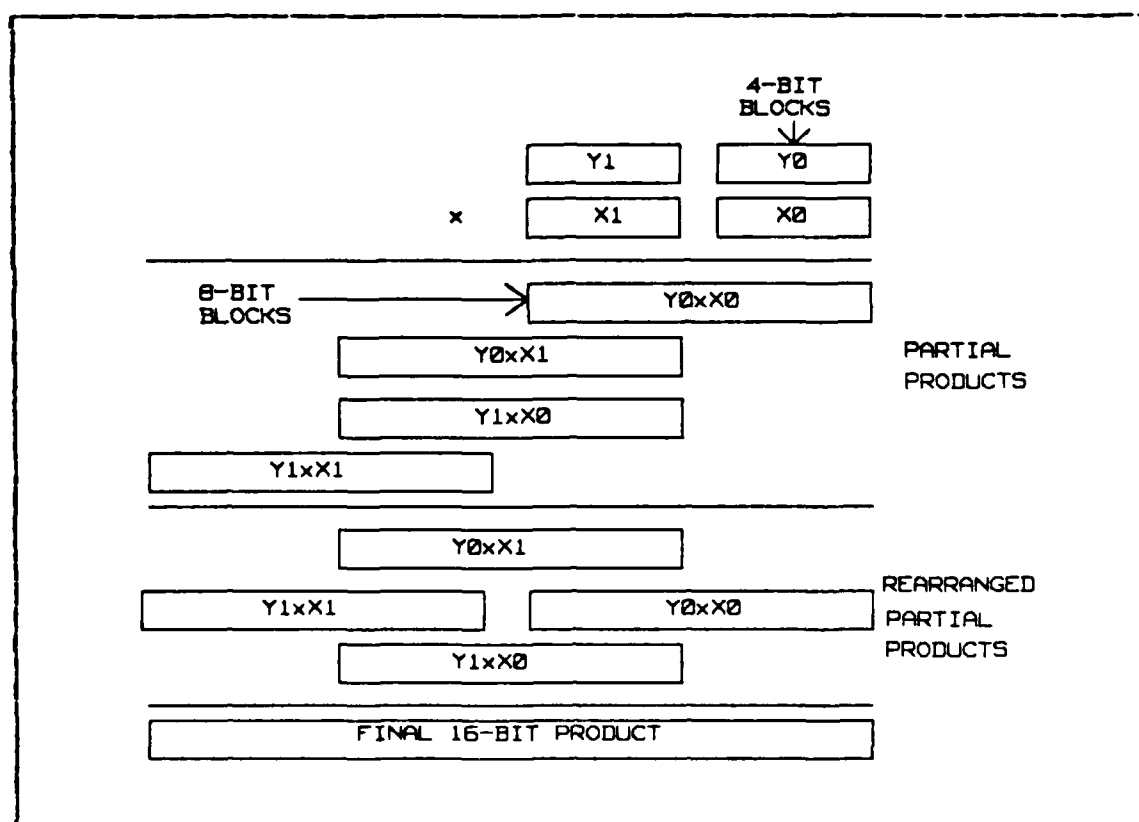
Figure 2.4   An 8x8 Multiplication Using ROMs.

(AND gate) method.   This method was chosen over the other
two because of its simple and regular implementation.
Booth's algorithm was rejected as a choice due to the
complex nature of the control signals that are required.
The ROM partial product generation method was not chosen
because it would require 16 ROMs of 65536 x 16 bits to
simultaneously generate the 16 partial products needed in a
16-bit multiplier.   Other possible combinations of different
size ROMs could also be used to generate the partial prod-
ucts, but due to chip area and feature size limitations
imposed by MOSIS the ROM method of generating partial prod-
ucts was rejected because it was not feasible to construct
on a single chip.

14
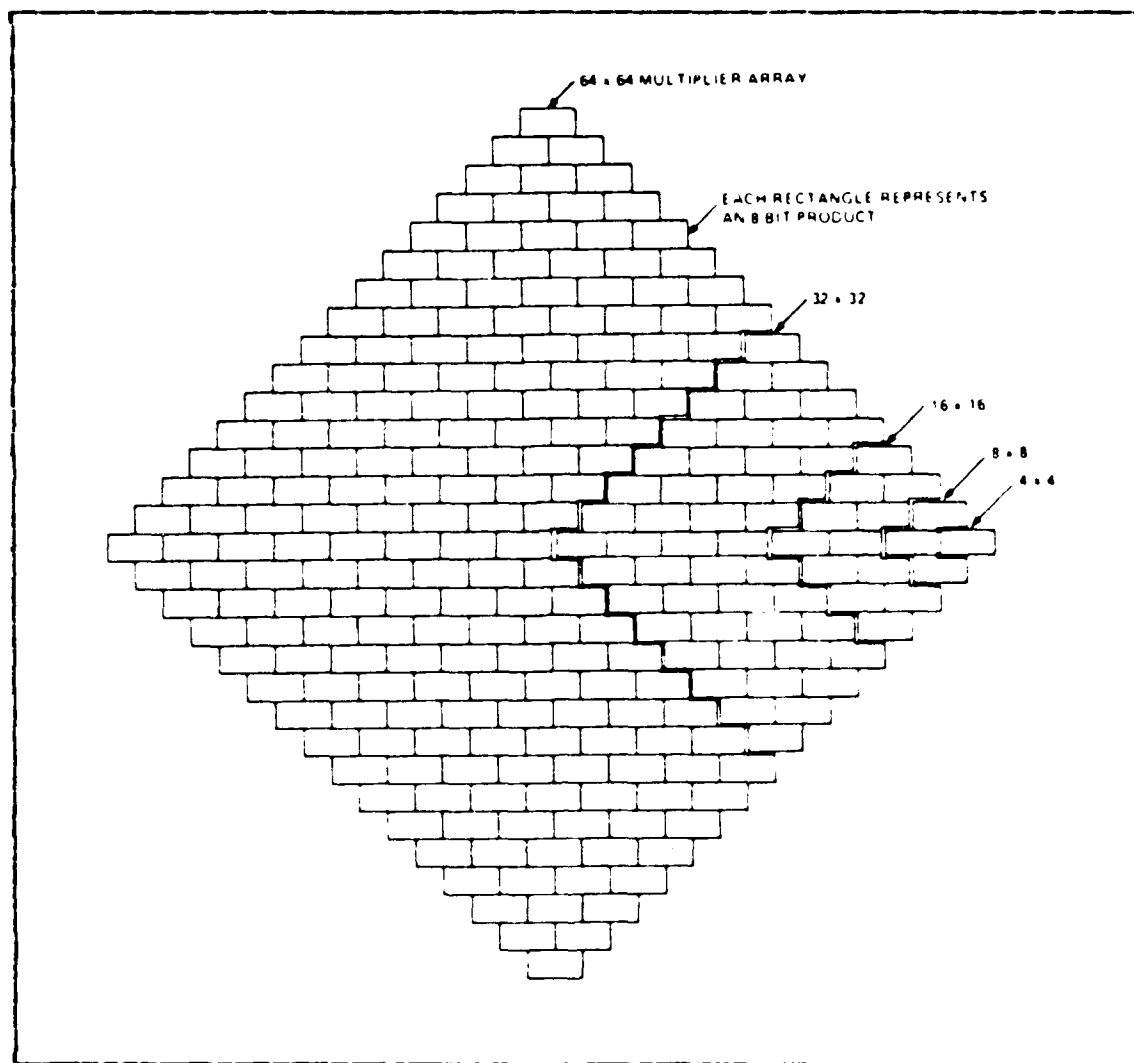
Figure 2.5   ROM Multiplier Weighted Position Structure.

2. Partial Products Reduction

Once the partial products are generated,   the next
step is to reduce the n partial products down to two.   One
technique that can be used to accomplish this is to utilize
3-input, 2-output full adders performing CSA in a Wallace
tree structure.

The partial products for   the 8x8 multiplication
represented by Figure 2.3 can be viewed as adjacent columns

15

TABLE I

Matrix Height for Partial Product Generation Methods

		MAX HEIGHT OF THE MATRIX							
	GENERAL	Number of Bits							
SCHEME	FORMULA	8	16	24	32	40	48	56	64
1 × 1 multiplier (AND gate)	n	8	16	24	32	40	48	56	64
4 × 4 multiplier (ROM)	$(n/2) - 1$	3	7	11	15	19	23	27	31
8 × 8 multiplier (ROM)	$(n/4) - 1$	1	3	4	7	9	11	13	15
Modified Booth's algorithm	$(n/2)$	4	8	12	16	20	24	28	32

of height h, where each column corresponds to all terms to the same power of 2, as shown in the Wallace tree structure of Figure 2.6.
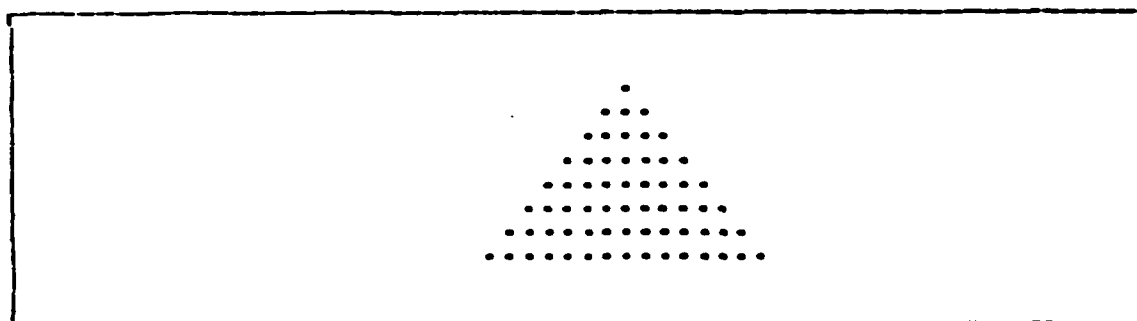
Figure 2.6   Partial Products in Wallace Tree Structure.

To reduce these columns of height h, CSA is used to reduce three dots of column height to two dots. These two output dots, which represent the familiar sum and carry outputs of a full adder, are placed in the next level of the tree structure in their appropriate power positions. In general, the number of required levels (L) of CSA required to reduce a Wallace tree structure of column height h to two is given by Equation 2.2 [Ref. 4: p. 139]. L can also be

viewed as the minimum number of full adder delays required to produce the pair cf column operands. For an 8x8 multiplication, the maximum column height is h=8. Thus, four levels of CSA are required as illustrated in Figure 2.7 [Ref. 4: p. 141].

$$L = \log_{1.5}\left(\frac{h}{2}\right)$$ (eqn 2.2)

Table II [Ref. 4: p. 139] shows the number of carry-save adder levels corresponding to various column heights.

3. Carry Look-Ahead Addition

The final step in this multiplication scheme is to sum the two remaining vectors created by the CSA reduction scheme discussed in the previous section. The major consideration in the choice of addition methods for the final summation is speed of operation. One method that significantly reduces the number of gate delays and increases the speed over ripple carry addition is carry lookahead (CLA) addition. Rather than give a full derivation of the CLA addition concept [Ref. 5: pp. 84-91], the basic operation is presented for the 32-bit CLA adder that is used in the final design implemented in this thesis.

Figure 2.8 represents the designed 32-bit CLA adder which can be thought cf as operating in three steps. First, the two input vectors X and Y to be summed are broken into 4-bit blocks. These blocks are routed into a circuit called a block P & G generator. The block P & G generator looks at each 4-bit block from X and Y to determine if a carry into the least significant bit position will propagate to the carry out of the most significant bit position of the block. The logic equations for these two signals, called block propagate (Pn) and block generate (Gn) respectively for bit position n, are given in Equations 2.3 and 2.4 for the nth bit position. Equations 2.3 through 2.15 are derived from [Ref. 5: pp. 84-91].

17

FIRST LEVEL
OF CSA

SECOND LEVEL
OF CSA

THIRD LEVEL
OF CSA

FOURTH LEVEL
OF CSA

LAST LEVEL:
CARRY-LOOK-
AHEAD
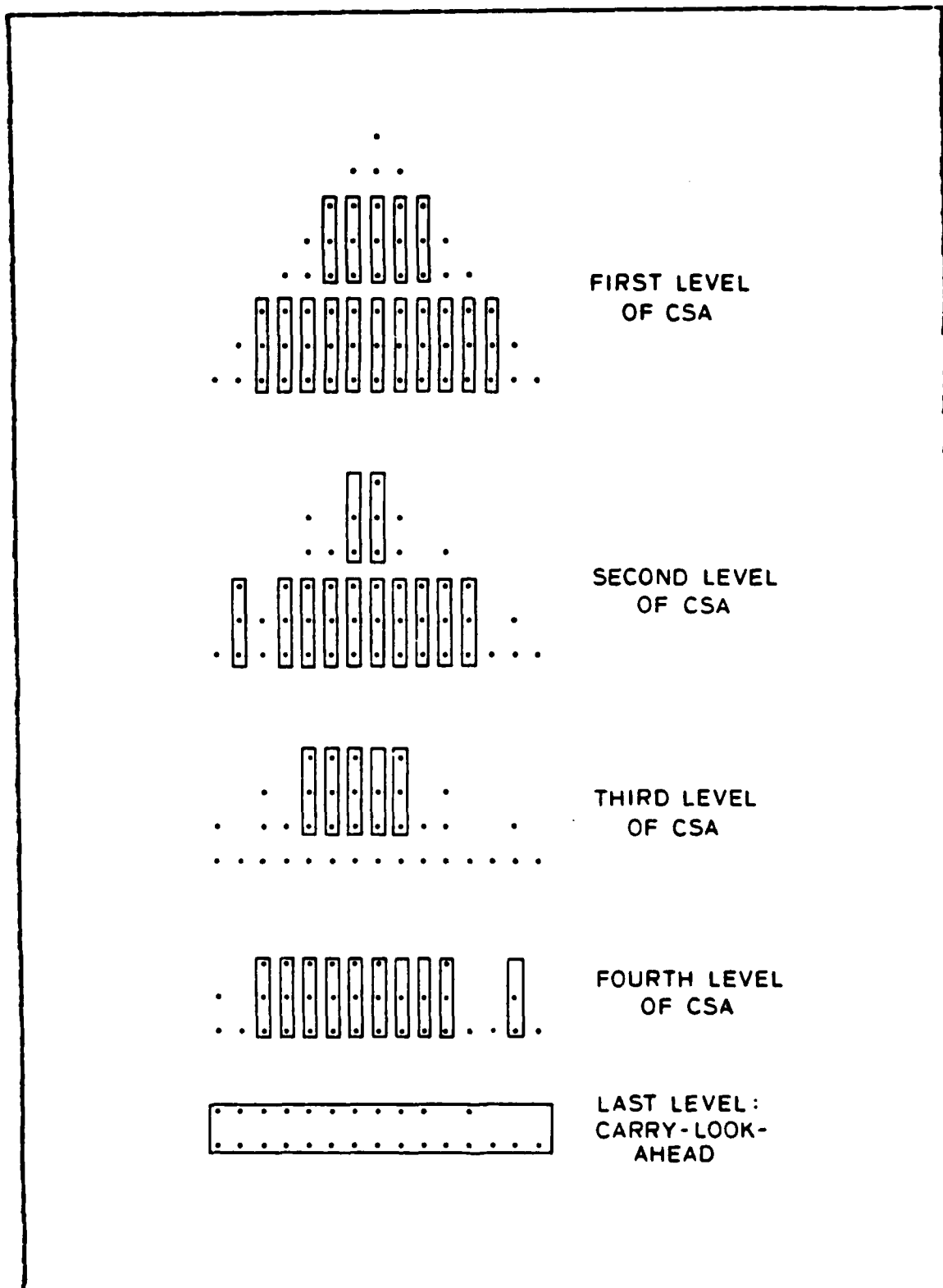
Figure 2.7   CSA Reduction for an 8-bit Multiplication.

18

TABLE II
Levels of CSA Needed vs. Maximum Column Height

Column Height (h)	Number of Levels (L)
3	1
4	2
$4 < n \leq 6$	3
$6 < n \leq 9$	4
$9 < n \leq 13$	5
$13 < n \leq 19$	6
$19 < n \leq 28$	7
$28 < n \leq 42$	8
$42 < n \leq 63$	9

$$P_n = \left(X_n + Y_n\right)\left(X_{n-1} + Y_{n-1}\right)\left(X_{n-2} + Y_{n-2}\right)\left(X_{n-3} + Y_{n-3}\right) \quad \text{(eqn 2.3)}$$

$$G_n = X_n Y_n + \left(X_n + Y_n\right)X_{n-1}Y_{n-1} + \left(X_n + Y_n\right)\left(X_{n-1} + Y_{n-1}\right)X_{n-2}Y_{n-2} \quad \text{(eqn 2.4)}$$

$$+ \left(X_n + Y_n\right)\left(X_{n-1} + Y_{n-1}\right)\left(X_{n-2} + Y_{n-2}\right)X_{n-3}Y_{n-3}$$

Next, the block P and G signals are input into a CLA unit that generates the true carry Cn out of the next least significant block C(n-1). For a 32-bit addition, two CLA units are required. The equations for the lower order CLA unit are given in Equations 2.5, 2.6, 2.7, and 2.8.

$$C_4 = G_3 + P_0 C_{in} \quad \text{(eqn 2.5)}$$

$$C_8 = G_7 + P_7 G_3 + P_7 P_3 C_{in} \quad \text{(eqn 2.6)}$$

$$C_{12} = G_{11} + P_{11}G_7 + P_{11}P_7 G_3 + P_{11}P_7 P_3 C_{in} \quad \text{(eqn 2.7)}$$

$$C_{16} = G_{15} + P_{15}G_{11} + P_{15}P_{11}G_7 + P_{15}P_{11}P_7 G_3 + P_{15}P_{11}P_7 P_3 C_{in} \quad \text{(eqn 2.8)}$$

Since in a multiplication of two numbers the carry into the least significant bit position is zero, the above four equations reduce to Equations 2.9, 2.10, 2.11, and 2.12.

$$C_4 = G_3 \quad \text{(eqn 2.9)}$$

19
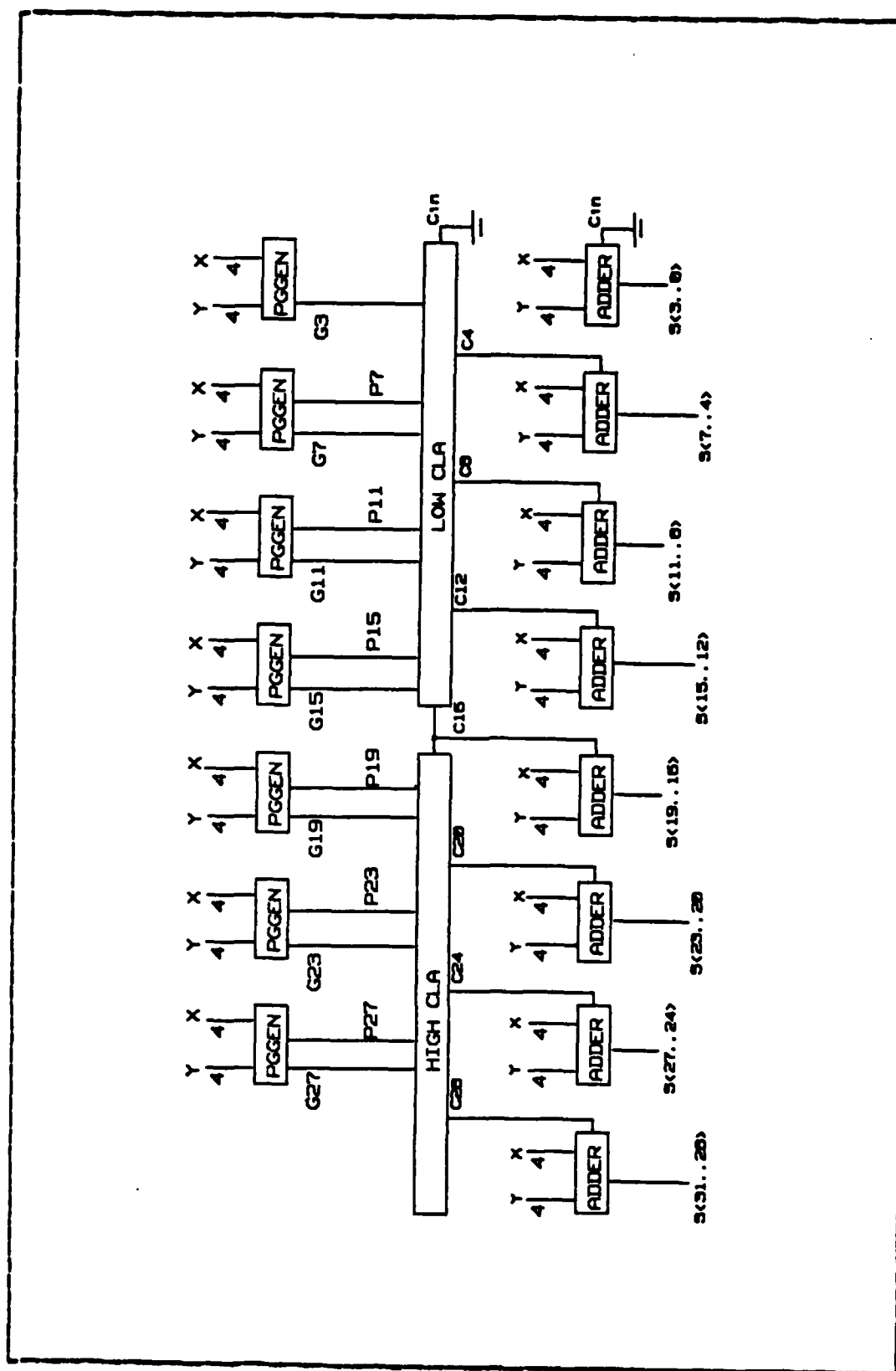
Figure 2.8   Block Diagram of a 32-bit CLA Adder.

$$C_8 = G_7 + P_7 G_3 \qquad \text{(eqn 2.10)}$$

$$C_{12} = G_{11} + P_{11}G_7 + P_{11}P_7 G_3 \qquad \text{(eqn 2.11)}$$

$$C_{16} = G_{15} + P_{15}G_{11} + P_{15}P_{11}G_7 + P_{15}P_{11}P_7 G_3 \qquad \text{(eqn 2.12)}$$

Similarly, the equations for the upper CLA unit are given as Equations 2.13, 2.14, and 2.15.

$$C_{20} = G_{19} + P_{19}C_{16} \qquad \text{(eqn 2.13)}$$

$$C_{24} = G_{23} + P_{23}G_{19} + P_{23}P_{19}C_{16} \qquad \text{(eqn 2.14)}$$

$$C_{28} = G_{27} + P_{27}G_{23} + P_{27}P_{23}G_{19} + P_{27}P_{23}P_{19}C_{16} \qquad \text{(eqn 2.15)}$$

Note that the carry out of the most significant bit is disregarded. This is because the result of multiplying two 16-bit operands yields only a 32-bit result.

Finally, the carry signals generated by the previous two steps are added in 4-bit block ripple carry adders with their appropriate slices of X and Y to form the 32-bit sum. Note that the carry out of each 4-bit ripple carry adder is disregarded, as it was generated and used previously.

C. PIPELINED ADAPTATION

In the previous section, the implementation of a parallel CSA multiplier was described. This method can logically be partitioned into stages for realization as a pipelined design.

In pipelining any design or algorithm, the basic objective is to introduce concurrency by taking the function to be performed and partitioning it into several subfunctions. The following properties [Ref. 6: p. 4] are important to consider when pipelining a design:

1. Evaluation of the basic function is equivalent to some sequential evaluation of the subfunctions.

2. The inputs for one subfunction come totally from the

outputs of the previous subfunction in the evaluation sequence.

3. Other than the exchange of inputs and outputs, there are no interrelationships between subfunctions.
4. Hardware can be developed to execute each subfunction.
5. The times required for these hardware units to perform their individual evaluations are usually approximately equal.

The hardware required to perform each subfunction of a pipeline is called a stage. At the output of each stage is a latch that is used to perform the actual exchange of operands between stages.

To partition the CSA multiplier into its stages, a logical division of the subfunctions to be executed must be determined. One method that initially may come to mind is to make the partial product reduction scheme using the Wallace tree structure as one stage of the pipeline and the CLA addition as a second stage. This was rejected because for a 16-bit multiply, the first stage would require six full adder delays and an AND gate delay before being ready to be latched. In the second stage, the CLA adder would require the delay for the P and G generation, the true carry generation in the CLA unit, and four full adder delays before being ready to be latched.

The next partitioning of subfunctions went one level further into defining each stage. The CLA adder was further subdivided into three subfunctions. The first stage performs the generation of the P and G signals based on the two 32-bit input vectors. The next stage uses the P and G signals generated in the previous stage to produce the true carry signals. In the third and final stage of the CLA adder, the 4-bit blocks are summed with their appropriate carry in signals generated in the previous stage to form the final product. In looking at the CLA adder portion, the

22

longest delay occurs in the final stage. This delay has a magnitude of 4 full adder delays and it is this figure that is used to partition the Wallace tree reduction scheme into stages.

For a 16-bit multiplication, the maximum height of the Wallace tree is sixteen as shown in Table I. This maximum height requires six levels of CSA addition (see Table II) before a column height of two is obtained to be input into the CLA adder. Also to be performed in this stage is the generation of each bit of the partial products through the use of AND gates. Starting at the beginning of the Wallace tree structure and keeping the stage delay at less than the four full adder delays of the CLA adder, the 1x1 multiply and three levels of CSA can be accomplished in the first stage of the pipeline. This leaves the next stage of the pipeline with the remaining three levels of CSA to perform before going into the 32-bit CLA adder for the generation of the final product. Figure 2.9 shows each stage of the pipeline and its subfunction. This pipelined structure is to be the one implemented in the final design of this thesis with adaptations to allow for the implementation of a two's complement multiplier.

```
              1x1 MULTIPLIERS

              1ST LEVEL CSA

              2ND LEVEL CSA

              3RD LEVEL CSA

                 LATCH

              4TH LEVEL CSA

              5TH LEVEL CSA

              6TH LEVEL CSA

                 LATCH

          BLOCK P & G GENERATORS

                 LATCH

                  CLA

                 LATCH

          4-BIT RIPPLE CARRY ADDERS

                 LATCH
```
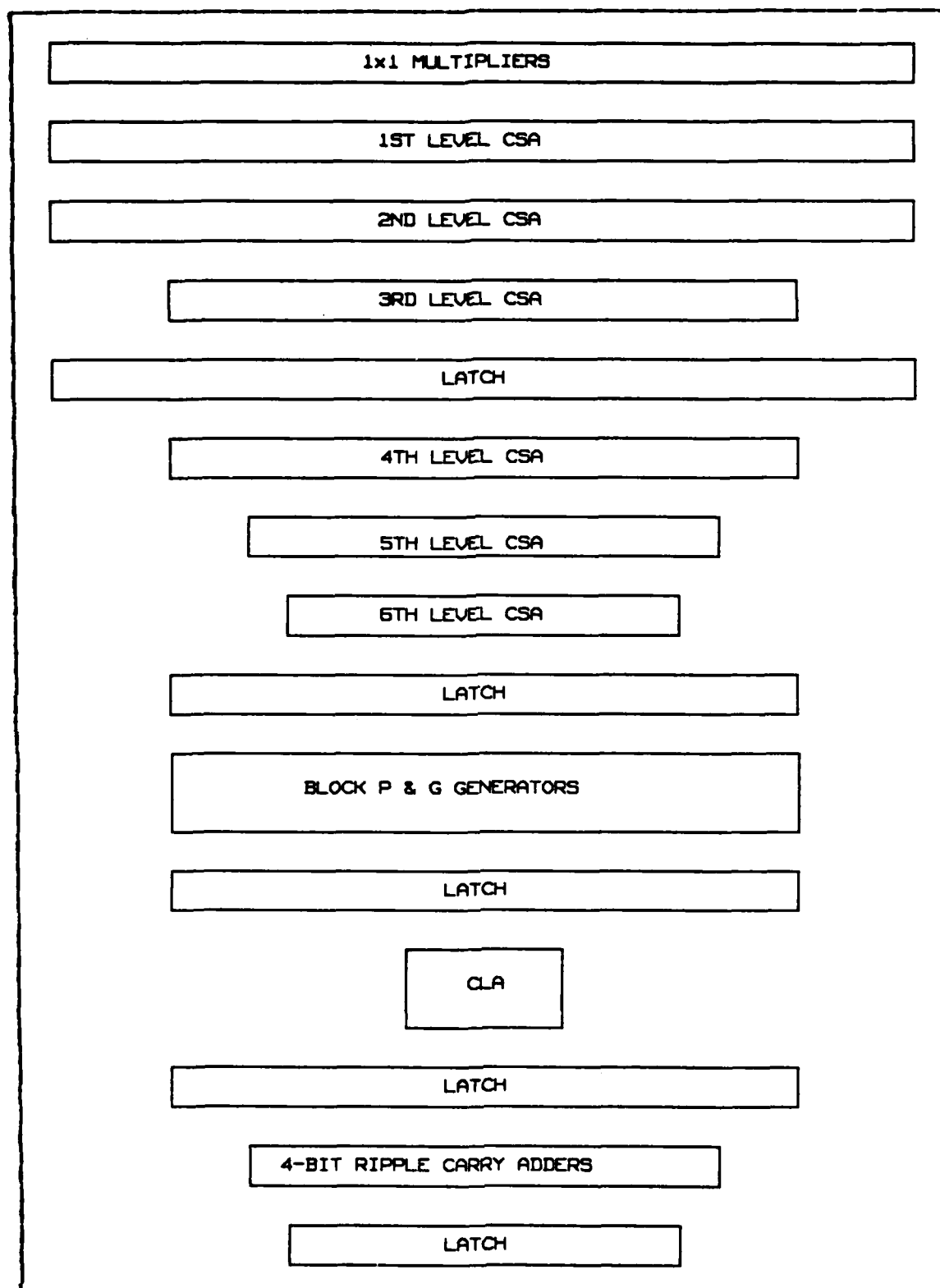
Figure 2.9   Pipelined CSA Multiplier.

III. DESIGN: 16-BIT TWO'S COMPLEMENT MULTIPLIER

A. TWO'S COMPLEMENT MULTIPLIER

1. Theoretical Architecture

The multiplication of two 16-bit signed numbers represented in two's complement form can be performed through the implementation of Equation 3.1 [Ref. 3] where n equals sixteen. In Equation 3.1, the notation b' denotes the one's complement of the multiplicand.

$$P = \sum_{k=0}^{n-2} 2^k a_k b - 2^{n-1} a_{n-1} b$$

$$= \sum_{k=0}^{n-2} 2^k a_k b + 2^{n-1} a_{n-1}\left(b' + 1\right)$$

$$= \sum_{k=0}^{n-2} 2^k a_k b + 2^{n-1} a_{n-1} b' + 2^{n-1} a_{n-1} \qquad \text{(eqn 3.1)}$$

Each partial product generated through the use of Equation 3.1 is summed with the remaining partial products as in the unsigned CSA multiplier discussed in the previous chapter with two exceptions. First, each partial product must have its most significant bit extended to the most significant bit of the final product. In the design used in this thesis for 16-bit operands, the most significant bit of each partial product must be extended to bit position 31. Second, the most significant bit of the multiplier must be added into bit position 15. This insertion of the most significant bit of the multiplier can also be accomplished by inserting it twice into the final summation at bit position 13 and once into each of the bit positions 14 and 15. This is done in the final design of this multiplier to keep the maximum column height to be input to the Wallace tree

25

reduction scheme at sixteen. Figure 3.1 demonstrates the use of this equation directly on the multiplication of two 4-bit two's complement numbers where n equals four.

```
    0111 = +7              0111 = +7
   x0101 = +5             x1011 = -5
 00000111               00000111
 0000000                0000111
 000111                 000000
 00000                  11000
 00000                  00001
 00100011 = +35         11011101 = -35

    1001 = -7              1001 = -7
   x0101 = +5             x1011 = -5
 11111001               11111001
 0000000                1111001
 111001                 000000
 00000                  00110
 00000                  00001
 11011101 = -35         00100011 = +35
```
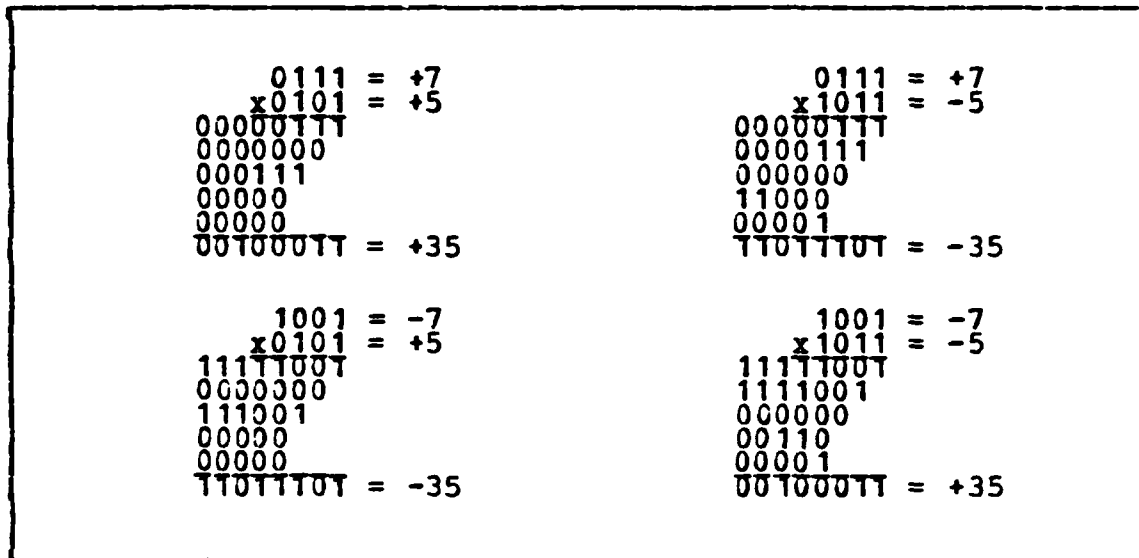
Figure 3.1   Two's Complement Multiplication.

$2^{15}a_{15}\underline{b}'$ $2^{15}a_{15}$ $2^{13}a_{13}\underline{b}$ $2^{0}a_{0}\underline{b}$
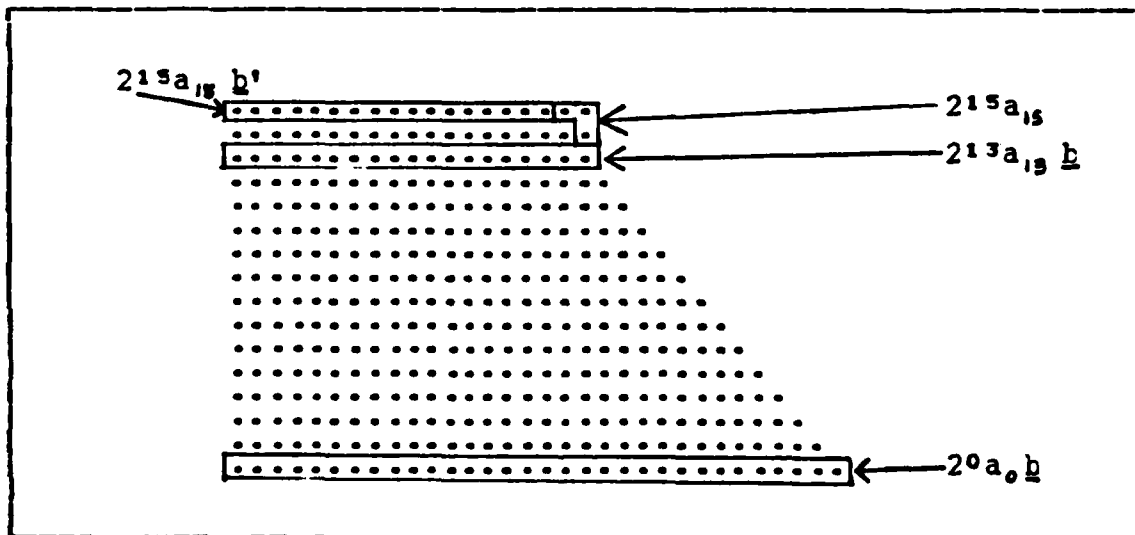
Figure 3.2   Input to Wallace Tree Reduction Method.

Figure 3.2 shows, in dot notation, the partial prod-
ucts generated with 1x1 multipliers using Equation 3.1 with
the two exceptions discussed above for a 16-bit two's
complement multiplication. It is this structure that is
input into the Wallace tree reduction scheme to be reduced
to a final maximum column height of two. Since the maximum
column height is sixteen for the 16-bit two's complement
multiplication presented in this thesis, six levels of CSA,
as shown in Figures 3.3 and 3.4, are required to decompose
this structure to a maximum column height of two. The
resulting two vectors generated by the CSA are then input
into the CLA adder presented in the previous chapter.

One interesting point to note is that the column
height for certain columns is only one. This is caused when
CSA is performed on three or less operands in a column and
no carry into that column is produced by the next lower
significant one. In these operand vectors, a zero is input
for the appropriate bit position into the CLA adder.

To perform this multiplication in a pipelined
manner, latches must be inserted at the end of each stage of
the pipeline as discussed earlier. Since the first stage
involves a 1x1 multiplication to generate the partial prod-
ucts and three levels of CSA, the first latch must be
inserted at the end of the third level of CSA. At this
point, 143 bits of data must be transferred to the second
stage. Therefore, the first latch is 143 bits wide.
Similarly, the second stage ends after the sixth level of
CSA is performed. This requires the second latch to be 57
bits wide. These 57 bits are then input to the CLA adder.
The third stage of the circuit generates the block P and G
signals. These signals and the 57 bits of the two CLA oper-
ands are then transferred to the fourth stage in a 70 bit
wide latch. The fourth stage uses the P and G signals to
generate the true carry signals to be used in the fifth and

1ST LEVEL OF CSA
(86 FULL ADDERS)

2ND LEVEL OF CSA
(93 FULL ADDERS)

3RD LEVEL OF CSA
(51 FULL ADDERS)
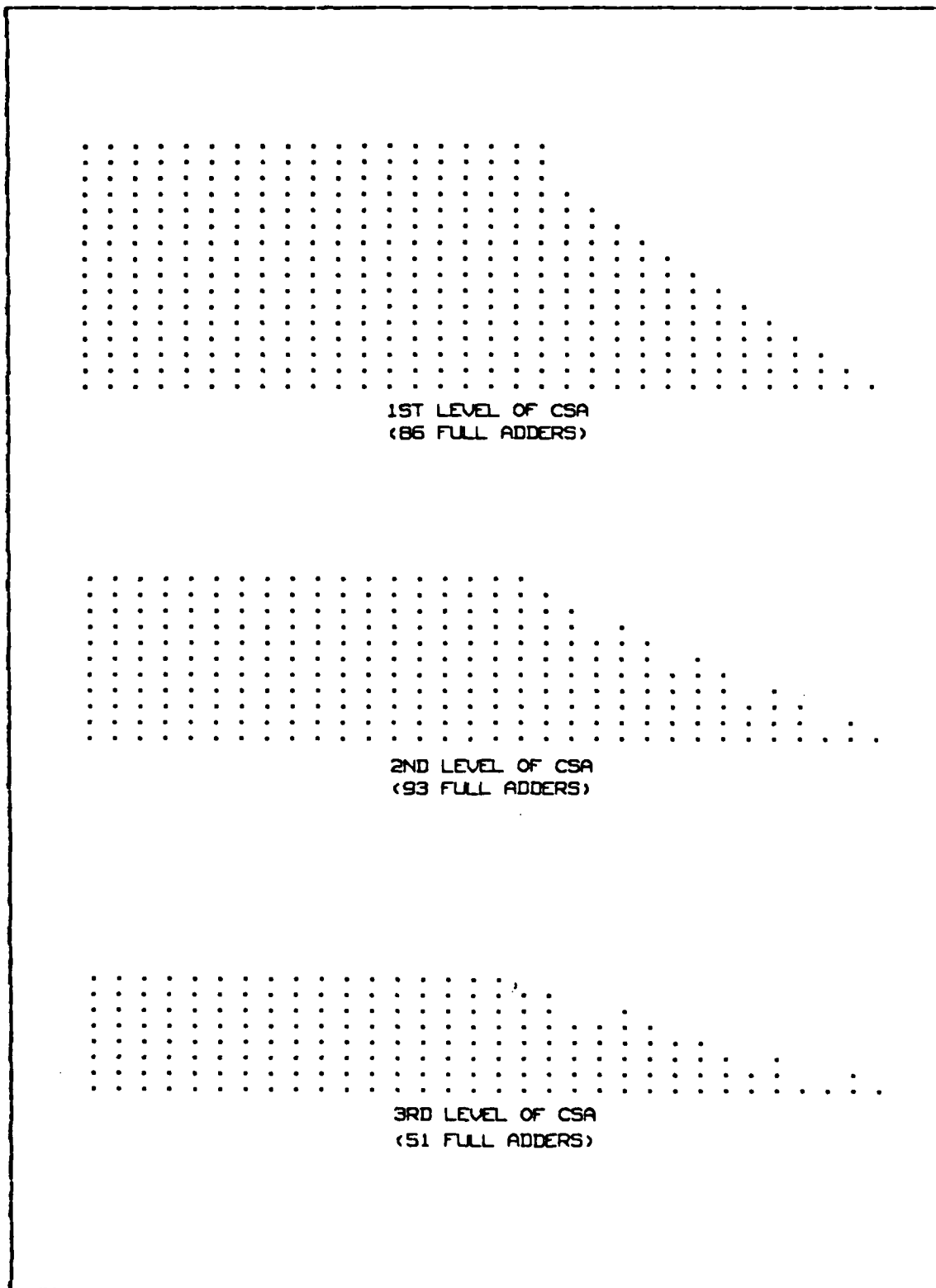
Figure 3.3   Partial Product Reduction Using CSA.
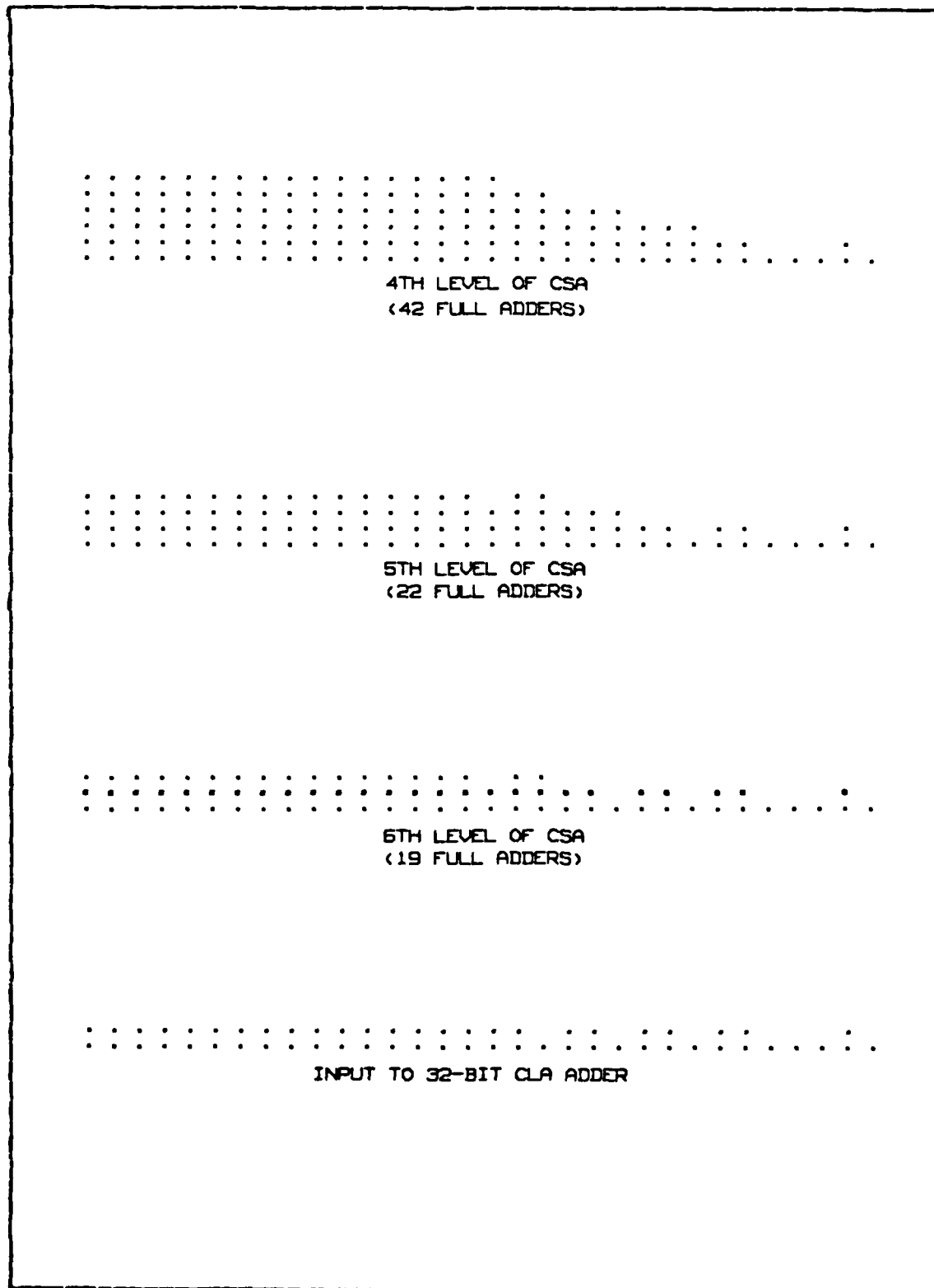
Figure 3.4   Partial Product Reduction Using CSA (cont'd.).

29

final stage.   This requires a 64 bit latch at its output to
hold the carry signals and the two CLA operand vectors.   The
final product appears at the output of the fifth stage and
is stored in a 32 bit wide latch so that latched outputs can
be provided to any subsequent circuits that this multiplier
may drive.

2. Actual Implementation

The initial floorplan for the circuit is shown in
Figure 3.5.   This floorplan closely follows the theoretical
implementation with two exceptions.

First, in a VLSI design, an AND gate used as a 1x1
multiplier is implemented with a NAND gate followed by an
inverter.   This active-high signal is then input to an
active-high input, active-high output full adder in the
first level of CSA.   Rather than construct these two circuit
elements in this manner, the actual implementation utilized
a NAND gate as the 1x1 multiplier driving an active-low
input, active-high output full adder.   Any signal generated
with a NAND gate as a partial product bit that is not used
in the first level of CSA is simply routed through an
inverter to convert it to an active-high signal for use in
subsequent levels of CSA.   This provided a reduction of 256
in the number of inverters to be constructed.

Second, the sign bits of each of the partial prod-
ucts must be extended to bit position thirty-one.   These
extended bits must also be added in the Wallace tree reduc-
tion of the partial products.   When these sign bits are
grouped for input to a full adder in the first level, up to
fourteen adders have the same three inputs.   Rather than
duplicate the adders which would increase power consumption
and usage of chip area, only one adder was used to calculate
the sum and carry inputs to the next level of CSA.   These
high fancut sum and carry inputs are then superbuffered to

30
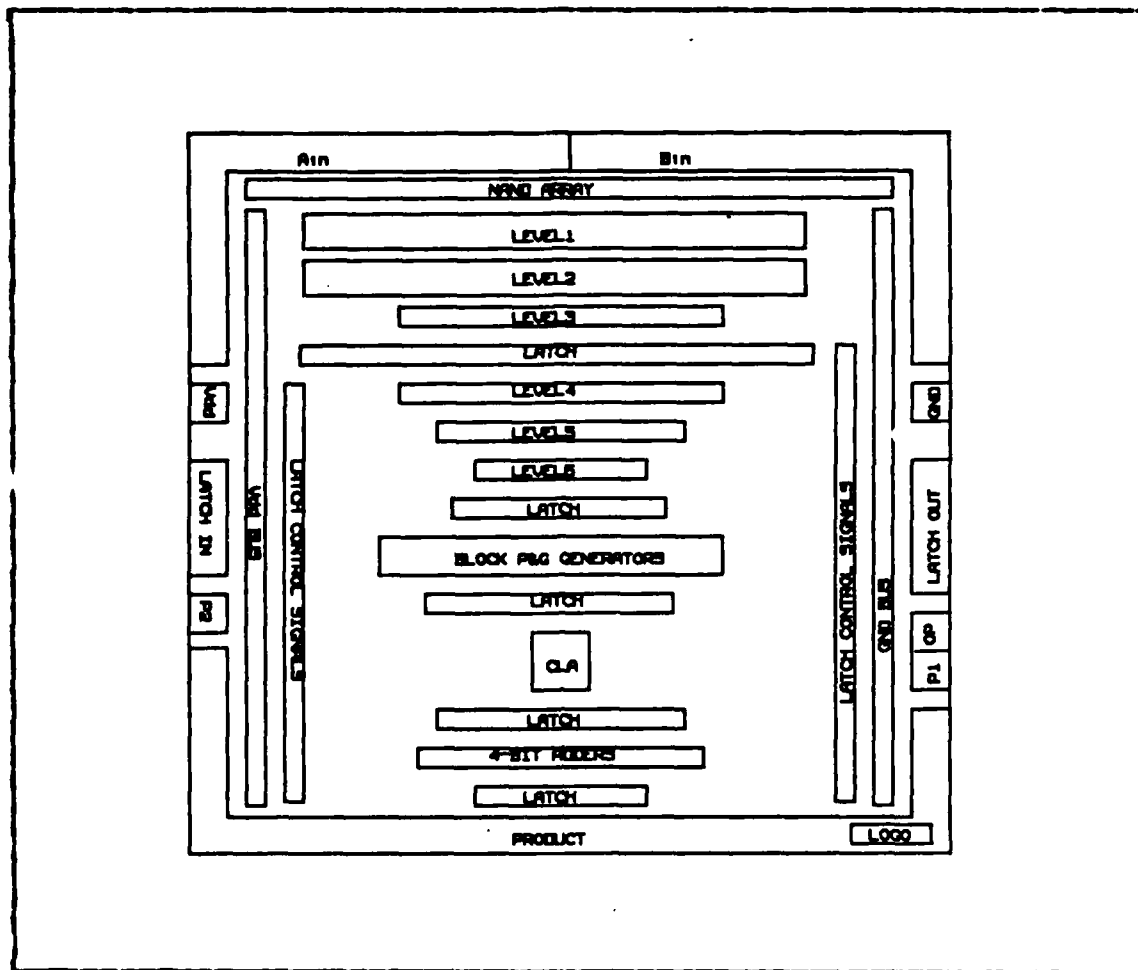
Figure 3.5   Initial Floorplan.

drive the second level of CSA.   This resulted in a savings
of thirty-five full adders not having to be implemented in
silicon.

The clocking of the circuit is   accomplished ky a
non-overlapping two-phase clock.   Both   phases are input to
the circuit through separate input pads.   An   additional
signal called OP is provided to allow for the implementation
of a level sensitive scan design (LSSD) [Ref. 7].   In a
LSSD,   the contents of the latches are either loaded in
parallel when OP is a high or serially shifted to an output

pad and serially loaded from an input pad when OP is low. This allows the contents of each of the first four latches to be examined to aid in the detection of fabrication errors or circuit malfunctions. The output latch is not serially loaded or shifted to an output pad because its contents are directly available at the output pads.

B. DESIGN TOOLS

Before the actual layout of a VLSI circuit can be undertaken, certain CAD tools are needed by the designer. First, a graphical layout editor is required to allow the designer to construct a VLSI circuit. Second, to allow for the implementation of complex logic functions, a PLA generator is desired. Next, the ability to employ a design rule checker on a layout is essential to insure that design rule violations do not unintentionally occur. Finally, tools that perform circuit simulation for logic, timing, and power consumption are useful in determining the proper operation of the designed circuit.

In the design of the 16-bit pipelined multiplier, the CAESAR layout editor [Refs. 7,8] was used as the basis for the layout of the entire chip. To facilitate the design of complex logic functions, EQNTOTT [Ref. 9] and TPLA [Ref. 9] were employed to construct complex programmed logic arrays (PLAs). LYRA [Ref. 9] was used to perform design rule checks on the circuit. Circuit simulation for logic, timing, and power were performed by ESIM [Refs. 2,9], CRYSTAL [Refs. 10,11] and PCWEST [Ref. 9] after a node extraction was performed using MEXTRA [Ref. 9].

The manuals for each of the CAD tools discussed above are available on the NPS Computer Science Department's UNIX operating system. To obtain an on-line copy of the manual for a specific design tool, issue the command

    % cadman <design tool name>.

32

To obtain a hardcopy of a certain CAD tool manual, issue the command

% cadman <design tool name> | lpr.

This command will send a copy of the normal CAD manual to the lineprinter.

1. **ECNTOTT**

ECNTOTT is a program which generates a truth table suitable for input to TPLA from a set of Boolean equations which define the PLA outputs in terms of its inputs. The equation syntax is

NAME = EXPRESSION;

where NAME is the output variable name and EXPRESSION is a Boolean equation in sum of products (SOP) form that represents the output variable in terms of its inputs. In the SOP expression, the & symbol denotes the logical AND, the | symbol denotes the logical OR, and the ! symbol preceeding an operand denotes the logical inversion. The input and output signal order, from left to right or top to bottom, as appropriate, can be controlled with the INORDER and OUTORDER commands.

2. **TPLA**

TPLA is a technology independent PLA generator that supports design rules in the following styles:

1. Mead-Conway NMOS with butting contacts, no buried contacts.
2. Mead-Conway NMOS with buried contacts, no butting contacts.
3. MOSIS 3 micron bulk CMOS.

It takes as its input the output of EQNTOTT and generates a PLA layout in the desired technology. The default output option is a CAESAR file. TPLA can provide inputs and outputs on either the same side (cis version) or on opposite sides (trans version) of the generated PLA. In addition, clocked inputs and/or outputs can be supported by TPLA through another option selection.

3. LYRA

LYRA is a design rule checker that operates on graphical files in CAESAR format. It can be invoked either interactively while editing a CAESAR file or on a CAESAR file and run in the background on the UNIX operating system. The interactive mode is discussed in earlier work done by Reid [Ref. 7]. In the background mode, LYRA is invoked by executing the command

% lyra filename.ca &.

This generates a file named CHECKPT which contains the names of all subcells of the design being checked that have completed a design rule check. If an error is found in the parent cell or any of its subcells, a file with the same name of filetype .ly is output to the user's current working directory. This file contains all error information and can be edited using CAESAR to view the errors for further correction. This mode of operation for LYRA provides an excellent means for design rule checking large designs that normally would take a long time in the interactive mode.

C. LAYOUT

Once the designer has determined the architecture to be implemented, the initial floorplan, and has mastered the CAD tools that are available, the next step in the design cycle

is to begin the layout of the actual circuit. One technique that is utilized in this design of a 16-bit pipelined multiplier is a form of the hierarchical design method. In this method, once the above three items are completed, the architecture is examined to look for some basic building blocks that could be designed and used repeatedly in the construction of the circuit. Upon examination of the architecture for the 16-bit pipelined multiplier, the four basic circuit elements that can be designed and iterated throughout the circuit are a full adder, a 4-bit block P and G generator, a CLA unit, and a 1-bit latch cell.

The full adder is the main element in both of the first two stages in the pipeline as well as a basic building block for the 4-bit ripple carry adders in the fifth stage. The first two methods of implementation that immediately arise are constructing an adder by using either discrete gates or a PLA generator such as TPLA. A third method [Ref. 12] that is possible is to use pass transistors in a selector logic circuit to generate the sum and carry bits that are conditioned on the three input bits to be added.

In choosing the adder to be implemented, two main considerations in the selection of the adder are its speed and power consumption. Both the discrete gate and the PLA adders have a higher static power consumption than the selector adder because they contain more depletion pull-up transistors than the selector adder. After simulation of these circuits for speed using CRYSTAL, it was found that the selector circuit, with a 14.7 nanosecond propagation delay, was faster than both of the other two by at least two nanoseconds. Therefore, the selector adder was chosen as one of the basic building blocks of the circuit. Figure 3.6 shows a circuit diagram of the selector adder used in the design of the 16-bit multiplier. Two minor drawbacks exist to the selection of this type of adder. When the output of

35

one adder drives the input of another, this is equivalent to
the output of a pass transistor driving an inverter. To
insure that the following adder inputs are driven to the
necessary voltage levels to operate properly, the input
inverters to each vertical selector rail must have a pull-up
to pull-down ratio of eight. Also, the selector rail that
provides the true signal to the circuit must pass through
two inverters. This prevents the output of a pass tran-
sistor in the previous adder from directly driving the gate
of a pass transistor in the current adder [Ref. 1: pp.
24-25].
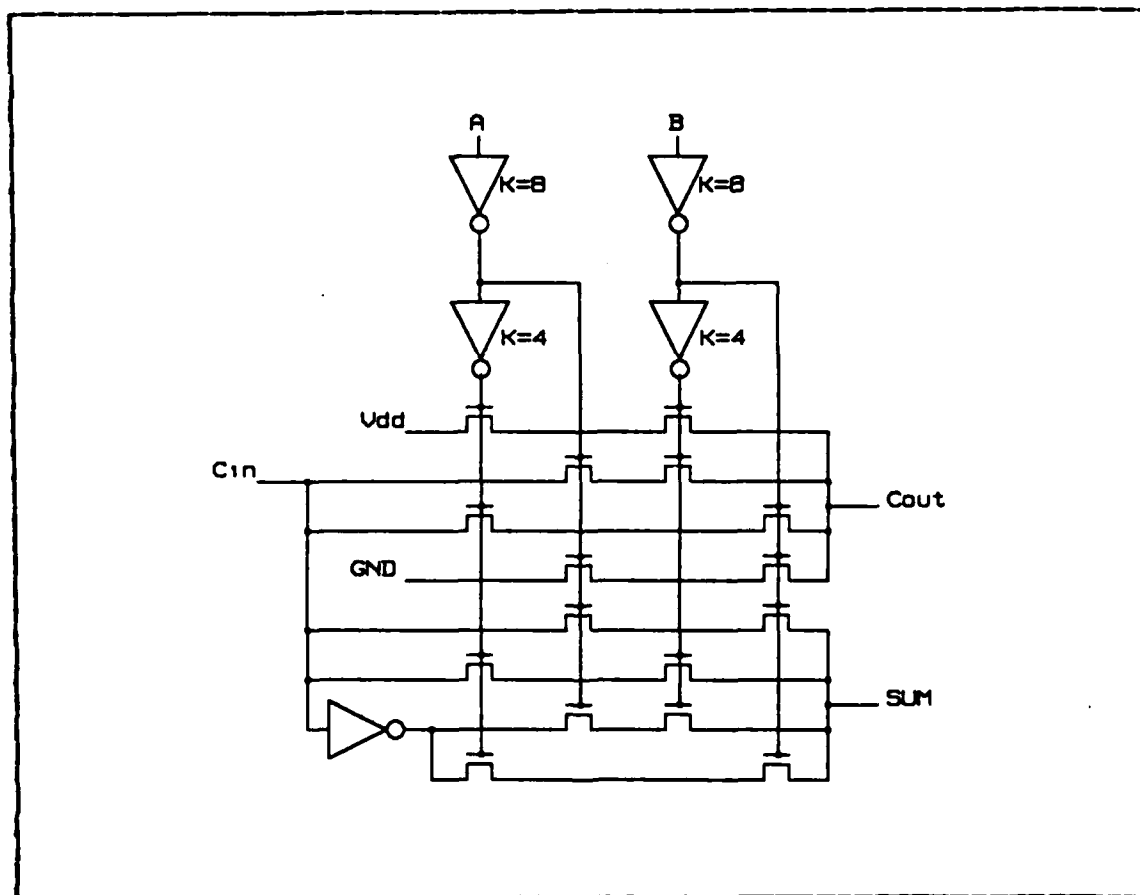
Figure 3.6   Selector Adder Circuit Diagram.

Both the 4-bit block P and G generator and the CIA unit
are complex logic functions well-suited for implementation
as PLAs. These two circuit elements are implemented by
inputting Equations 2.3 and 2.4 (for the P and G generator)
and Equations 2.9 to 2.15 (for the CLA unit) into EQNTOTT.
The output of EQNTOTT is then piped to TPLA to generate the
actual CAESAR files for the PLAs. Since data flows into one
side and out from the opposite side of each stage, the trans
version of the PLAs was constructed.

The last building block of the circuit to be designed is
the 1-bit latch cell. Since a LSSD is an important
criterion for designing the 16-bit multiplier, the 1-bit
latch cell must be able to be loaded either in parallel
along the data path or in serial from an adjacent latch
cell. This function is under control of the OP signal.

To minimize the area consumed by the latch, a dynamic
latch composed of a pair of inverters coupled by pass tran-
sistors was selected. As in the adder circuit, a pull-up to
pull-down ratio of eight is needed for the inverters because
they are driven by pass transistors. Figure 3.7 shows the
circuit diagram of the 1-bit latch cell as implemented. The
operation of the latch cell is as follows. For normal oper-
ation (OP=1), the NORMAL signal is high and the SHIFT signal
is low during PHI1. Data appearing at the DATA IN port
drives the first inverter. When PHI1 falls, the gate of the
first inverter retains the logic value of DATA IN in its
gate capacitance. When PHI2 rises, this data drives the
second inverter which effectively transfers the data to DATA
OUT and the next stage. For a shifting operation (OP=0),
the NORMAL signal is low and the SHIFT signal is high. Data
appearing at the LATCH IN port, which connects to DATA OUT
of the next latch cell to the left, charges the gate capaci-
tance of the first inverter. The pass transistor transfers
the data to the second inverter on PHI2 as in a normal

37
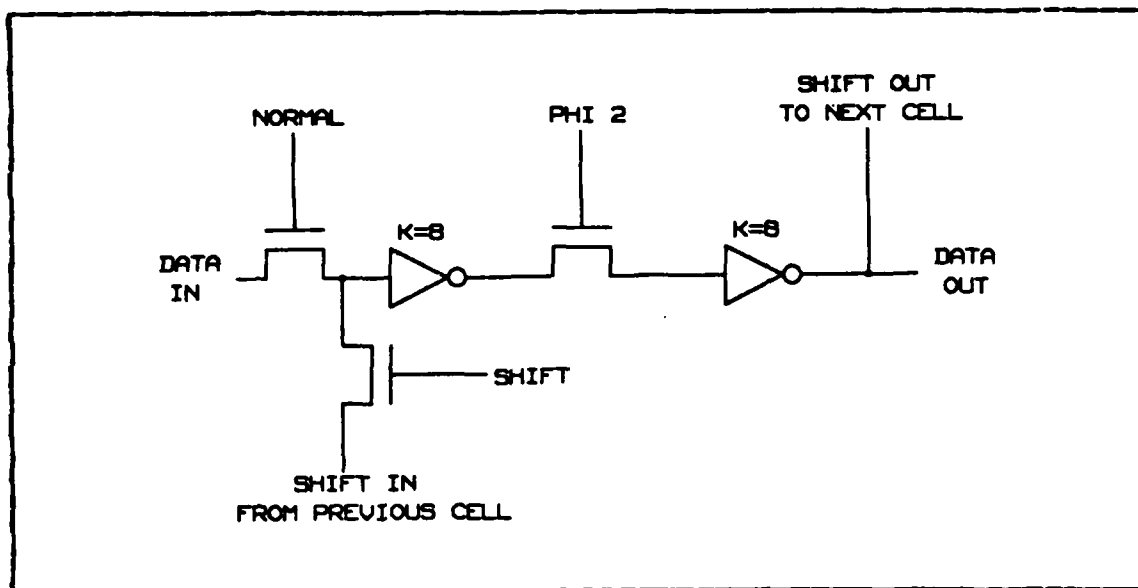
Figure 3.7   1-bit Latch Cell.

operation.   This effectively shifts the data from the LATCH
IN port   to the LATCH   OUT port in   one cycle of   the clock.
Figure 3.8 shows the circuitry to   condition PHI1 with OP to
generate the NORMAL and SHIFT signals used above.
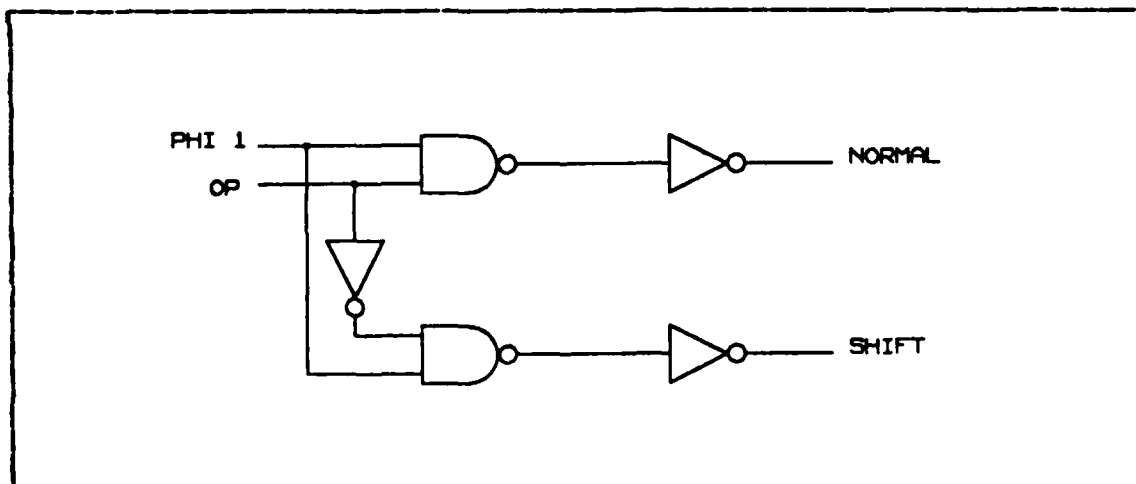
Figure 3.8   Generation of the Control Signals.

38

Once these four basic building blocks are designed, each
stage of the pipeline and its latch is developed out of the
appropriate subcells. Next, the internal routing of signals
within a stage is accomplished through the use of a wire
list. Then the five stages of the circuit are wired
together to form the core of the design. Finally, all that
remains to be done is to connect this core design to a frame
to allow adequate interfacing for the packaging process.

This routing of signals both within the core of the
design and to the frame is an extremely time consuming task
that requires as much time, effort, and planning as the
design and layout of all the major components. The addition
of an automatic router would be a welcome addition to any
designer's CAD toolbag.

The design frame is composed of a pad set that was
obtained from MOSIS. These pads were specifically designed
for fabrication at 1.5 microns per lambda. A copy of these
pads is located in the file

/vlsi/berk83/lib/pads15.cif

and associated documentation can be found in the file

/vlsi/berk83/doc/pads15.

Both of these files are located in the NPS Computer Science
Department's VAX11-780 running the UNIX operating system.

Numerous repetitions of the design - rule check - rede-
sign cycle occurred before a final design was obtained.
Using LYRA for the design rule check on a large design such
as the 16-bit multiplier requires approximately 1000 CPU
minutes. When the UNIX system is heavily loaded, this
results in a turn-around time on the order of two to three
days. Figure 3.9 depicts the final design of the entire
chip. Each of the six levels of CSA are shown as level1
through level6. The latches are labelled latchxx where xx
is the appropriate number of bits in the latch. The block P
and G generators are designated PG and the CLA unit is

39

simply shown as CLA. The 4-bit ripple carry adders are
shown as ADD. Three blocks not previously discussed are
labelled AMP. These are control line drivers that drive the
high fancut NORMAL, SHIFT, and PHI2 signals to each of the
latches. These drivers are composed of the same circuitry
used by the output pads to drive off chip loads.
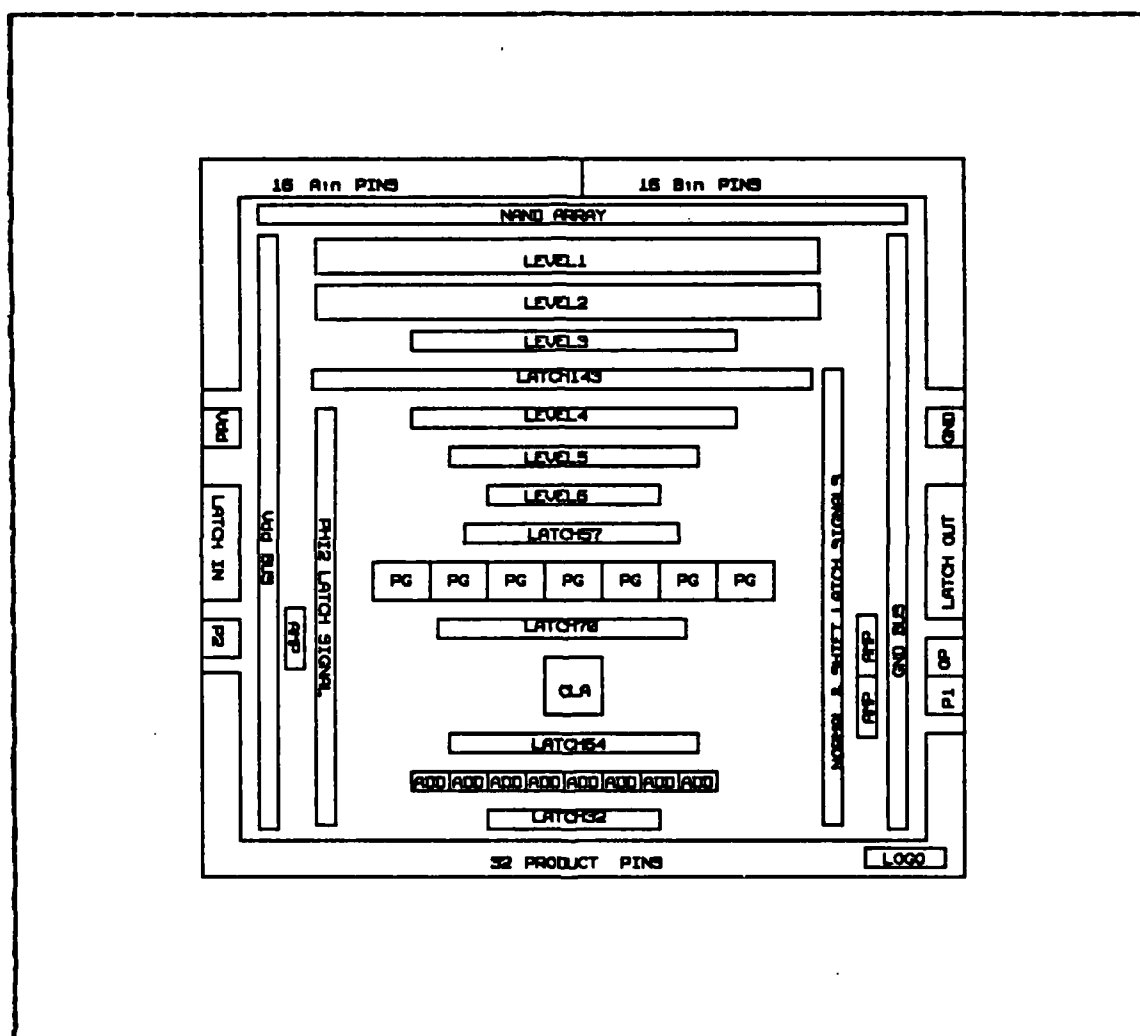
Figure 3.9   Final Chip Floorplan.

The actual plots cf each of the four building blocks and
the final circuit layout are contained in Appendix A. These
plots were generated using the program CIFPLOT [Ref. 9].

C. DESIGN VALIDATION

The next step in the design cycle is to functionally validate the chip's operation before it is sent to MOSIS for fabrication. This will give the designer a high degree of certainty that the chip operates logically as desired with an approximate power consumption and at a certain maximum frequency of operation.

Before these three items can be accomplished, two preliminary steps must be accomplished. First, the CAESAR file must be edited to label the nodes and a Caltech Intermediate Format (CIF) file generated. For the purpose of performing design validation using CAD tools, the scale of centimicrons per lambda must be an even multiple of four. This prevents round-off errors in the resultant CIF file. Since the final design is to be fabricated at lambda equals 1.50 microns, 152 centimicrons per lambda is used. Second, the CIF file must be passed through the MEXTRA program using the command

    % mextra -o filename.cif &

so that a node extraction is performed on the circuit. On large files, it is extremely useful to run this program in the background mode as shown by the & in this command. A large CIF file such as the one for the 16-bit multiplier can take up to thirty minutes of CPU time to run. When the UNIX system is heavily loaded, this requires eight to ten hours of real time. The output files are directly compatible with the CAD simulation tools to be used.

1. Logical Simulation

The first step in any design validation process is to determine if the circuit functions as it was designed to. Today, as the complexity of VLSI designs increases, the

number of possible inputs goes up tremendously. For example, to exhaustively test just the normal operation of the 16-bit multiplier would require each possible combination of the 16-bit multiplier and multiplicand inputs. The number of possible combinations of the vectors a and b is

$$(2^{16})^2 = 2^{32} = 4,294,967,296.$$

The ESIM logic simulator is the CAD tool to be used for checking operation of the 16-bit multiplier. If a vector pair is input only once, without regard to order, and at an estimated rate of two test vector pairs simulated per minute, this would require

4,294,967,296 vectorsx1 day/2880 tests=1.49x10⁶ days.

This is equivalent to:

$$4,294,967,296 \text{ vectors} \times 1 \text{ day}/2880 \text{ tests} = 1.49 \times 10^6 \text{ days}.$$

This amounts to over 4085 years required to perform an exhaustive test.

Therefore, seven representative pairs of test vectors were selected for simulation to determine if the circuit operates correctly. Exhaustive testing is not possible, but most possible errors would be revealed by these few, carefully chosen test vectors. These seven test vectors are:

1. +143 x +27
2. -143 x +27
3. +143 x -27
4. -143 x -27
5. +1123 x +891
6. -1123 x +891
7. -32768 x -32768

These vectors were designed to test as large a number of subcircuits as possible. The first four vector pairs test the basic architecture for the correct implementation of the algorithm represented by Equation 3.1. The positive/

negative and negative/negative test vector pairs also test the CIA adder's ability to produce a proper sum over the entire thirty-two bit width. The next two vector pairs test the ability of the CSA in the Wallace tree reduction scheme to produce a correct result in the upper sixteen bits of the product. The last test vector is the largest negative number representable in 16-bit two's complement form. Further simulation with additional test vectors would increase the confidence of the designer in the ability of the circuit to properly simulate a 16-bit two's complement multiplication prior to fabrication.

Once the read-in of the .sim file by ESIM is completed, the initialization of the circuit, the defining of watched nodes, and describing the clock cycles must be accomplished before any simulation is performed. Rather than do this each time ESIM is entered, a macro file was created that is called at the beginning of each session. This file is called init_esim and is shown in Figure 3.10 for the 16-bit multiplier. The input vectors for the two operands are represented as ain and bin. The resultant product vector is shown as phigh and plow representing the upper and lower 16-bits of the 16-bit product, respectively. The latch input and output signals are represented as the vectors latchin and latchout where the leftmost bit corresponds to the first latch and the rightmost bit to the fourth latch.

After initialization of the circuit by executing the init_esim macro, at each clock cycle the seven test vector pairs previously defined are input in sequential order. In each case, on the fifth clock cycle after introduction of a test vector, the correct product appeared at the output pads phigh and plow. This demonstrates that the circuit can properly multiply two 16-bit two's complement operands to yield a 16-bit result with the result dependent only on the

43

```
I
I
I
I
I
w op
W ain a15 a14 a13 a12 a11 a10 a9 a8 a7 a6 a5 a4 a3 a2 a1 a0
W bin b15 b14 b13 b12 b11 b10 b9 b8 b7 b6 b5 b4 b3 b2 b1 b0
W latchin l1_in l2_in l3_in l4_in
W phigh p31 p30 p29 p28 p27 p26 p25 p24 p23 p22 p21 p20 p19 p18 p17 p16
W plow p15 p14 p13 p12 p11 p10 p9 p8 p7 p6 p5 p4 p3 p2 p1 p0
W latchout l1_out l2_out l3_out l4_out
K phi1 01000 phi2 00010
h op
s
```

Figure 3.10   Initialization Macro for ESIM.

inputs to the circuit five clock cycles prior.   The results
of this logic simulation are contained in Appendix B.

The serial shifting of the latches was simulated and
used to generate the intermediate results discussed in the
next chapter.   This also proved to logically operate as
expected, thus giving the designer a high degree of confi-
dence that the circuit operates as desired.

2. Timing

The CRYSTAL VLSI timing analyzer is used to test for
the worst case propagation delay in the circuit.   Each phase
of the clock in both a normal and shifting operation is
checked for a critical path that is defined to be within one
percent of the worst case propagation delay. These critical
paths determine the maximum clock speed at which the circuit
can properly operate.   The worst delays found are discussed
for each phase of the clock.

On the rising edge of an externally applied phi1, the longest propagation delay occurs from the input pads until the data is stored in the first inverter of the stage 1 latch. This delay is found to be 558.82 nanoseconds. This long delay can be attributed to the two high fanouts that occur in the data path of the first stage. The first is a fanout of sixteen that occurs at each input pad to the input of the sixteen NAND gates used as 1x1 multipliers. The second is a fanout of fourteen that occurs at the end of the first stage where the full adder cells that correspond to the extended sign bits are distributed to drive full adders in the second stage .

When phi1 falls, it takes 89.11 nanoseconds for the latch cells to turn of their input pass transistors and isolate the data so it may be transferred during phi2. This fall time corresponds to the separation time between phi1 and phi2 when both clock phases are low.

Once a rising clock edge is applied to phi2, it takes 98.26 nanoseconds for the pass transistors in the latch cells to turn on and charge the second inverter. To complete the transfer of data, these pass transistors must be disabled by the falling of phi2. This corresponds to the minimum separation between the phi2 and phi1 clock phases and is found to be 64.28 nanoseconds.

Figure 3.11 depicts the minimum clock cycle for the 16-bit multiplier as determined by CRYSTAL. This equates to a maximum overall clock frequency of 1.234 MHz. The results of the CRYSTAL timing analysis are contained in Appendix B.

3. Power Consumption

DC power requirements for the 16-bit multiplier are determined through the use of the CAD program POWEST. POWEST looks for pullup transistors and determines a total count of these devices. Using a reference power consumption

45

PHI 1

558.82

89.11

64.28

PHI 2

98.26

NOTE: ALL TIMES IN NANOSECONDS.
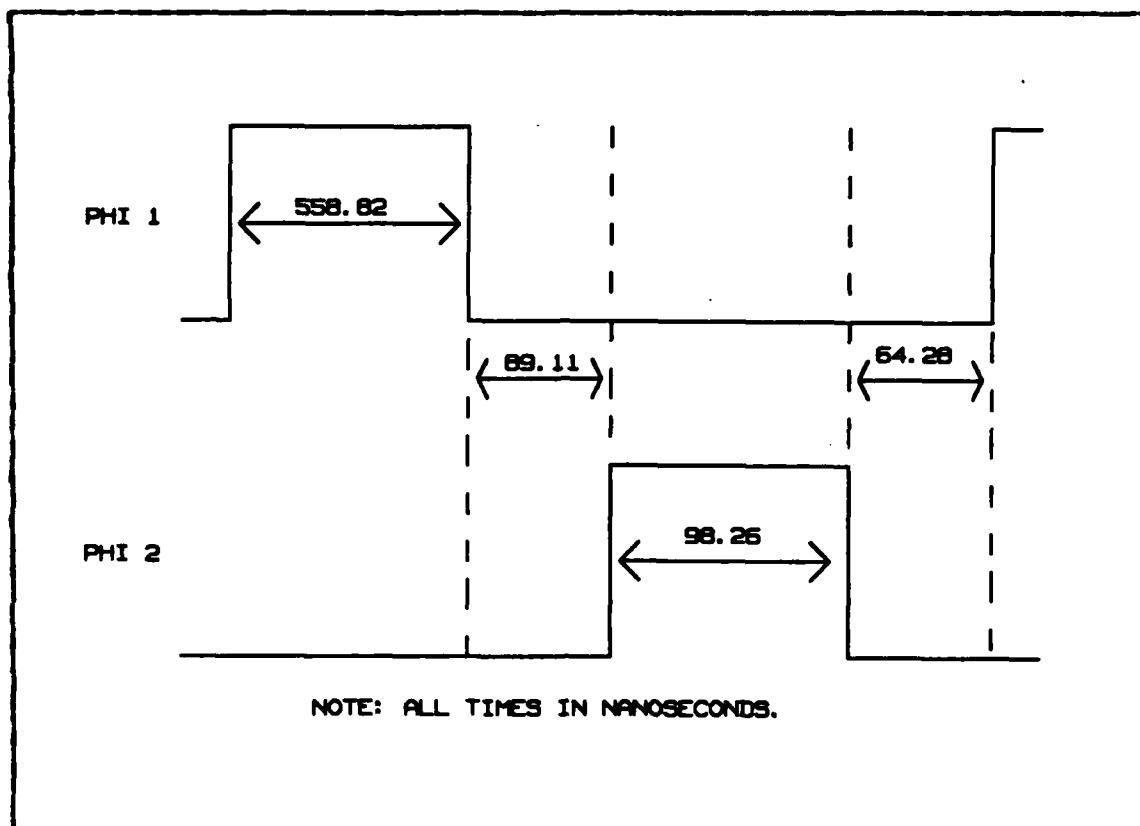
Figure 3.11   Minimum Clock Cycle Parameters.

for pullup transistors of certain sizes and types, it obtains a maximum estimate of power consumed by assuming all pullups are on at the same time. The average power consumption is determined by assuming that only half of the pullups are on at a given time.

For the 16-bit multiplier, the ma imum DC power consumption is found to be 3.177 Watts with a average power consumed of 1.983 Watts. The results of the LOWEST simulation are found in Appendix B.

46

IV. TEST PLAN

As stated earlier, the use of the logic simulator ESIM, the CRYSTAL timing analyzer, and POWEST will give the designer a high degree of confidence that the circuit designed will perform as desired. Once the circuit has been fabricated and received from MOSIS, it must be tested to insure that fabrication and/or bonding errors did not occur. Preliminary work done by Carlson on a 16-bit pipelined multiplier indicates that errors in fabrication and/or bonding do actually occur. In this chapter, a test plan for the verification of power consumption, correct logical operation, and maximum speed of operation is presented.

A. IDENTIFYING INPUT AND OUTPUT PINS

After fabrication, the chip will come back packaged in an 84 pin square grid package with 21 pins on each side. Since only 77 pins are used in the 32-bit multiplier, it is imperative that the pin to pad connections are accurately known. To do this, one must properly orient the chip. Close examination of the chip will reveal the logo "GC ARMY" located between the GND and Vdd rails that run around the perimeter of the chip. Place this logo in the southeast corner as shown in Figure 4.1. Using this logo as a landmark, proceed clockwise around the chip starting on the southern edge.

Along the southern edge are twenty-one output pads that are used for a portion of the product. Representing the product as p31...p0 where p0 is the least significant bit, the southern edge contains signals p6 through p26 as one moves from east to west. The western edge is made up of

47

Vdd

GND

A9 A8 A7 A6 A5 A4 A3 A2 A1 A0 B15 B14 B13 B12 B11 B10 B9 B8 B7 B6 B5

A10
A11
A12
A13
A14
A15

Vdd

L1_IN
L2_IN
L3_IN
L4_IN

PHI 2

P31
P30
P29
P28
P27

CORE

OF

DESIGN

B4
B3
B2
B1
B0

GND

L1_OUT
L2_OUT
L3_OUT
L4_OUT

OP
PHI 1

P0
P1
P2
P3
P4
P5

P26 P25 P24 P23 P22 P21 P20 P19 P18 P17 P16 P15 P14 P13 P12 P11 P10 P9 P8 P7 P6 LOGO
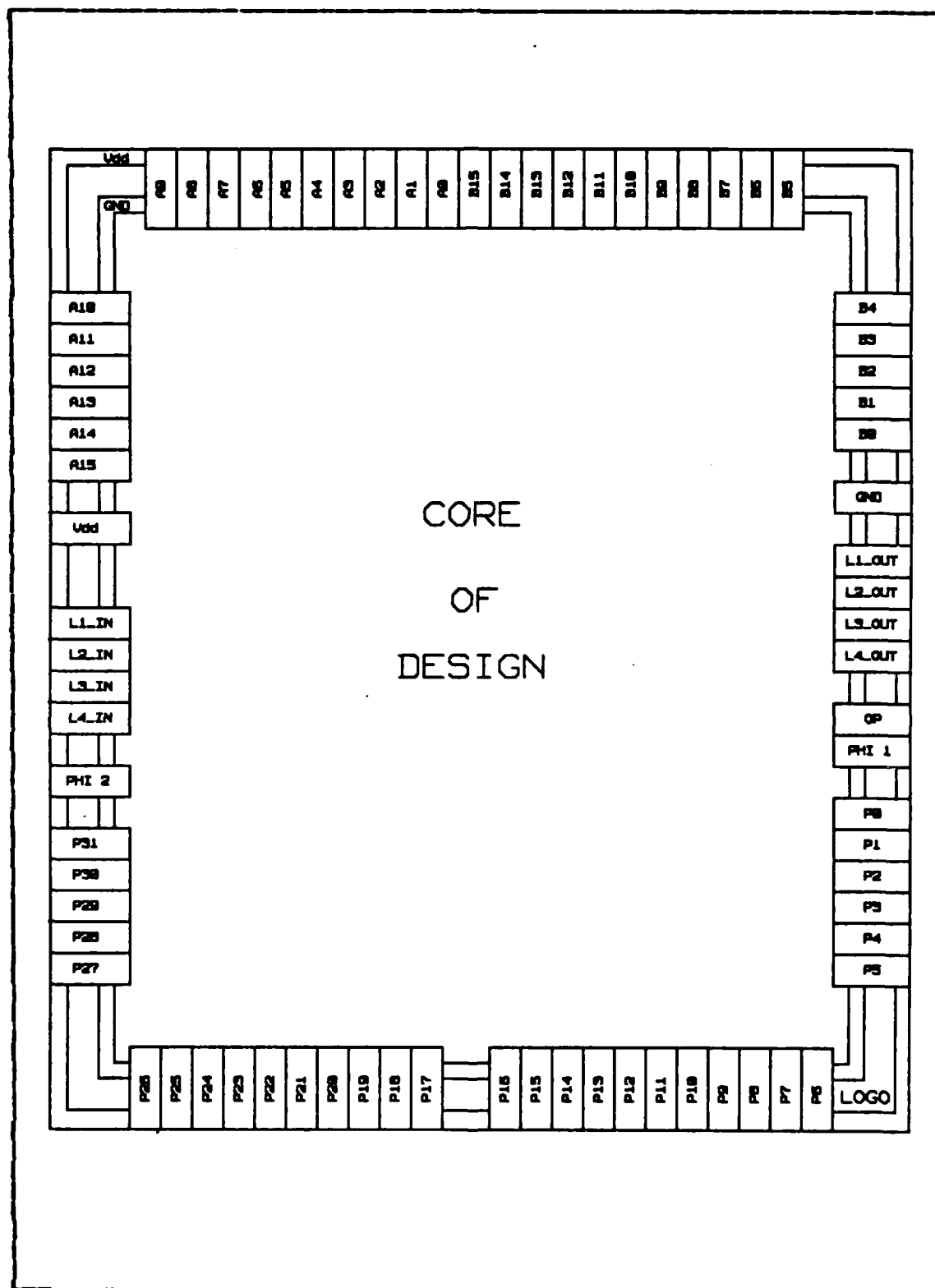
Figure 4.1   Pad Identification.

48

five output pads and twelve input pads. Moving from south
to north, the first five pads are p27 through p31. The next
pad is the phi2 clock input followed by the four latch
serial inputs for latch 4 through latch 1. Then comes the
Vdd pad followed by the six most significant bits of the
multiplier a15 through a10. Moving west to east along the
northern edge, the remainder of the multiplier inputs a9
through a0 and the eleven inputs of the multiplicand b15
through b5 are encountered. Along the eastern edge going
from north to south, the remainder of the multiplicand pads
b4 through b0 are found followed by the GND pad. Next are
the four latch serial outputs for latch 1 through latch 4.
Next are the OP and phi1 inputs which are followed by the
lower six bits of the product vector p0 through p5. This
should complete the circuit around the chip and leave one
back at the logo. Extreme care must be exercised when
tracing the fine wires from the bonding pads to the pins,
especially along the east and west edges where the number of
pins is greater than the number of bonding pads.

To power the chip +5 volts DC should be applied to the
Vdd pad and 0 volts to the GND pad. All inputs should use
Vdd to represent a logic 1 and GND for a logic 0. The
outputs use the same levels as the inputs to represent the
two logic levels. To measure the outputs, they should be
connected to a device with a high input impedance.
According to the documentation for the pads, the output pads
are designed to drive approximately two TTL loads, but may
require a pullup resistor to obtain a full Vdd output level.

B. POWER CONSUMPTION

The simplest of the three tests to perform is to check
the static DC power consumption of the circuit. Once input,
output, and supply pins are properly connected, this can be

49

accomplished by inserting a milliammeter into the Vdd supply line and measuring the nuber of amperes the circuit is drawing. This value multiplied by the +5 volts of the power supply will give an approximate average DC power consumption. This figure should be in the vicinity of the 1.983 Watts predicted by POWEST.

C. TESTING FOR LOGICAL OPERATION

Since exhaustive testing of the 32-bit multiplier is virtually impossible, the same seven test vectors that were used in ESIM should be utilized to verify correct operation. In addition, other random vector pairs should be tested for correct operation in the circuit. At this point, speed of operation is not a ccncern and the clock frequency should be reduced by a magnitude of approximately ten from that predicted by CRYSTAL. This will insure that propagation delays do not beccme a factor in determining logical correctness.

First, the vector pairs should be applied one at a time and a minimum of five clock cycles completed with OP at a logic 1. At the end of the fifth clock cycle, the output should represent the correct product for the input pair. This will at least insure that the chip performs a 32-bit two's ccmplement multiplication. This should be done for each cf the seven test vector pairs that were used in ESIM. Next, each of the seven test vector pairs should be applied every clcck cycle. After a delay of five clock cycles, the correct results should appear at the output during phi2 of each cycle of the clock. This establishes the fact that the chip can multiply in a pipelined manner.

To determine if the latches can serially operate as designed, known sequences should be applied at the inputs with the OP pin at a logic 0. Since the latches that are

50

output to the four latch output pads are all of different
lengths, the output of this operation will occur at
different times for each pin. For latch 1, latch 2, latch 3
and latch 4, the input sequence will start appearing at the
appropriate output pin after 143, 57, 70 and 64 clock
cycles, respectively.

If any of the test vectors fail, the intermediate latch
results of each vector pair can be shifted to an output pin
for examination. This can provide an excellent aid in
locating circuit faults. The intermediate latch values and
the final product outputs for each of the seven test vector
pairs are found in Appendix C.

D. TESTING FOR MAXIMUM SPEED

The third and final test to be performed on the chips
that pass the logic function testing is to determine the
maximum frequency at which they will operate correctly. To
accomplish this, the duration of the time that phi1 and phi2
are high and the two interphase times when phi1 and phi2 are
low should be separately reduced until an incorrect product
is generated. This should be done with each of the seven
test vectors until a minimum time is found for each of these
four clock parameters. Then the worst case for each of
these parameters over all seven test vectors can be called
the minimum clock parameters for the 32-bit multiplier. The
maximum overall clock frequency for the chip is then just
the reciprocal of the sum of the four minimum clock
parameters.

V. FULL CUSTOM VS. SILICON COMPILER DESIGN

One of the main advantages of using a silicon compiler is that it provides an extremely fast transition time from the initial architecture to the final layout of the design. This author estimates that the total time to actually generate the design of the 8-bit multiplier by Carlson [Ref. 2] using the MacPitts silicon compiler was less than 24 man-hours. Theoretically, at the end of this time, a functionally correct layout is generated. Later work done by Froede [Ref. 11] on this compiler has proven that MacPitts does not always generate a correct layout. In comparison, the time consumed in the design of the 16-bit multiplier presented in this thesis is estimated at over 750 man-hours.

This design turn-around time advantage of using a silicon compiler for chip generation allows the designer a great degree of freedom to explore possible different architectures to solve a problem and actually see the results in silicon. This freedom is not enjoyed by the full custom designer whose architecture must be thoroughly researched and optimized prior to the layout of the actual chip. If this is not the case, a tremendous loss of valuable man-hours occurs when the redesign of a chip's basic architecture must be undertaken.

The use of a silicon compiler is not without its disadvantages though. Three of the main areas that a silicon compiler generated chip is at a disadvantage are:

1. density of transistors.
2. speed of operation.
3. power consumption per transistor.

To make a specific comparison, an 8-bit multiplier generated by the MacPitts silicon compiler available at NPS was compared with the full custom multiplier of this thesis. The following sections discuss the three main areas listed above. They are preceeded by a discussion of the two circuit architectures that are to be compared.

A. FUNCTIONAL ARCHITECTURE

The architecture of the 16-bit multiplier has already been thoroughly presented in the previous two chapters. In summary, the chip performs a 16-bit two's complement pipelined multiplication on 16-bit operands with a latency of five cycles of a two phase clock. The circuitry for this chip is designed using a minimum feature size of 3.0 microns and is wholly contained on one integrated circuit.

The multiplier generated by the MacPitts silicon compiler performs an 8-bit multiplication on unsigned 8-bit operands with a latency of eight cycles of a three phase, five segment clock. It uses the basic add-and-shift algorithm for the basis of its architecture. Due to the limitations in chip dimensions, pin count, and minimum feature size imposed by MOSIS at the time the chip was fabricated, this chip was designed with a minimum feature size of 4.0 microns. It requires the cascading of two identical integrated circuits to perform an 8-bit multiplication.

Additionally, the 16-bit multiplier employs a LSSD technique that allows the contents of each of the four intermediate latches to be serially examined to aid in the detection of circuit fabrication errors. The MacPitts multiplier does not employ this technique and determing fabrication and/or design errors is extremely difficult, if not impossible, to perform by examining just the chip outputs. A LSSD technique could possibly have been included

53

in the MacPitts design, but if included the maximum chip area defined by MOSIS may have been exceeded.

B. CHIP AREA AND DENSITY

Since both VLSI circuits are designed with different minimum feature size, to provide a fair basis for comparison of the two designs the 16-bit multiplier is normalized to a 4.0 micron feature size. Figure 5.1 shows the resultant .log file from the MEXTRA node extractor for both the 8-bit and 16-bit multipliers. This file contains the chip dimensions in microns and the number of transistors in the circuit.

Window: 0 676600 0 602400
801 depletion
1612 enhancement
1398 nodes

MacPitts 8-bit Multiplier.

Window: -600 919350 -600 789300
3914 depletion
11962 enhancement
8503 nodes

Custom 16-bit Multiplier.

Figure 5.1   MEXTRA .log Output.

The size shown in Figure 5.1 for the 16-bit multiplier is based on a 1.5 minimum feature size. This results in

chip dimensions of 9199.50 by 7899.0 microns. By current MOSIS limitations, the maximum chip dimensions are 9200.0 by 7900.0 microns. Therefore, at lambda equal 1.5 microns the overall design is within one micron or less of the maximum allowed by MOSIS. Normalizing the circuit dimensions to a 4.0 micron minimum feature size, the 16-bit multiplier consumes an area 12,260.0 by 10,532.0 microns. By comparison, the MacPitts generated 8-bit multiplier occupies an area 6766.0 by 6024.0 microns. The MacPitts chip consumes approximately one-third of the area of the hand-crafted multiplier.

The other main point of interest that deals with the physical characteristics of the chip is its transistor density or number of transistors per square micron. For the normalized 16-bit multiplier, Figure 5.1 shows a total of 15,876 transistors. This yields a transistor density of 1.23×10^{-4} transistors per square micron. For the MacPitts multiplier, the MEXTRA node extraction found a total of 2,413 transistors. This gives a transistor density of 5.92 $\times 10^{-5}$ transistors per square micron. One interesting point to note is that the MacPitts compiler found eighty-four more transistors on the 8-bit multiplier than the MEXTRA node extractor did [Ref. 2]. One possible explanation for this difference is that MacPitts generates some unusual transistor structures that were unrecognizable by MEXTRA.

C. POWER CONSUMPTION

One area that is becoming more and more important with the increasing number of transistors per chip that is being created by improved technology is the static DC power dissipation of a VLSI circuit. For the purposes of providing comparisons, the CAD program PCWEST is used as the basis for reference.

For the 16-bit multiplier, the average DC power consumption is found to be 1.983 Watts with a maximum power usage of 3.177 Watts. Using POWEST on the 8-bit multiplier yielded an average DC power consumption of 0.352 Watts and a maximum power usage cf 0.667 Watts. Appendix B contains the results of the POWEST runs on both of the designs. The MacPitts silicon compiler also outputs an estimate it makes of the maximum power consumed by a circuit. For the 8-bit multiplier, this value is 0.407 Watts. This value is over thirty-five percent less than the POWEST maximum value.

One way to possibly compare the power consumption for the two designs is to determine a power consumed per transistor figure. Using the maximum POWEST values for both designs yields 2.00×10^{-4} Watts per transistor for the 16-bit multiplier and 2.77×10^{-4} Watts per transistor for the 8-bit multiplier. The difference between these two figures can be primarily attributed to the following. The MacPitts multiplier uses nine two input NAND gates to generate the full adders used in each stage. The custom multiplier uses a selector adder composed primarily of pass transistors which consume no DC static power. This results in an overall lower power consumption per transistor for the 16-bit multiplier when compared to the 8-bit multiplier.

D. SPEED OF OPERATION

As discussed earlier, CRYSTAL determined that the maximum clock frequency for the 16-bit multiplier is 1.234 MHz. MacPitts generated designs use a different clocking scheme than the two phase, ncn-overlapping clock presented by Mead and Conway [Ref. 1: p. 65]. It uses a three phase, five segment overlapping clock to generate the control signals for each latch in the pipeline. For a full discussion cf the MacPitts clocking scheme and how to use the

CRYSTAL timing analyzer on a MacPitts design, the reader is referred to work done by Froede [Ref. 11]. The timing analysis was performed on the MacPitts multiplier in accordance with this document and the worst-case CRYSTAL timing results are contained in Appendix B.

The overall minimum clock period for a CRYSTAL design is found by adding the worst stage propogation delay that occurs during the first two segments of the clock to the last three clock segment delays. For the 8-bit multiplier, the longest stage is the first. The critical path is found to run from the input pads, through the Weinberger array, and then through eight full adders cascaded in series to perform one summation of the partial products in the add-and-shift algorithm. This delay was found to be 4838.89 nanoseconds. The sum of the individual times for the clock signals to travel from the input pads to the latch cells during the last three segments of the clock is 207.14 nanoseconds. This results in an overall minimum clock period of 5046.03 nanoseconds and a maximum clock frequency of 198.176 KHz. The high propogation time in the first stage of the circuit is due primarily to three things. First, high resistance polysilicon is utilized for the long data runs. Second, no signals are buffered in any way to provide an improved signal sourcing capability to help combat the high fanouts and long data runs. Third, an 8-bit ripple carry adder is utilized to sum two partial products in every stage of the pipeline. Each 1-bit full adder in an 8-bit ripple carry adder is composed of nine NAND gates. The carry in between each full adder in the ripple carry adder is not routed directly, but is routed over a long polysilicon wire which also contributes to the high critical path delay.

57

E. SUMMARY

Table III summarizes the results for the comparison of the hand-crafted design and its silicon compiler generated counterpart. The results are as expected with the custom design having a six-fold increase in maximum speed, a thirty-eight percent decrease in power consumption per transistor, and a doubling of chip density over the MacPitts design. The true advantage of the MacPitts silicon compiler is in its ability to provide extremely rapid design turnaround time versus a hand-crafted design. As research continues into the area of silicon compilation and improvements are made to existing compilers, they may someday become the powerful and useful tool that they have the potential to be.

TABLE III
Summary of Comparison Statistics

PARAMETER	CUSTOM MULT	MACPITTS MULT
SIZE OF OPERAND INPUTS	16 bits	8 bits
DIMENSIONS (microns)	12266 x 10532	6766 x 6024
DENSITY (transistors/micron^2)	1.23×10^{-4}	5.92×10^{-5}
STATIC DC POWER (Watts)		
POWEST		
AVERAGE	1.983	0.352
MAXIMUM	3.177	0.667
MACPITTS		
MAXIMUM	NA	0.407
POWER/TRANSISTOR (Watts)	2.00×10^{-4}	2.766×10^{-4}
MAXIMUM FREQUENCY (KHz)	1234.0	198.176
DESIGN TIME (man-hours)	750	24

VI. CONCLUSION

In this thesis, the application of carry-save addition
to a 16-bit two's complement multiplication and its imple-
mentation as a pipelined VLSI design have been presented. A
comparison between this hand-crafted design and an 8-bit
unsigned multiplier was developed. This comparison coupled
with the experience gained in the actual design and computer
simulation of the multiplier leads to the following conclu-
sions and recommendations.

A. DESIGN OF THE MULTIPLIER

If the design of the multiplier were to be undertaken
again, three changes to the circuit would be desirable.
First, the incorporation of a static latch would be
attempted provided a feasible design that would fit into the
limited available chip area could be developed. A static
latch would insure that data remains valid and not be
discharged from the inverter's gate capacitance if too slow
a clock is applied. Second, the high fanout from the latch
control drivers would be divided into a tree structure. At
its termination points would be smaller, more efficient
drivers that would drive a fanout not greater than five.
Third, improvements to the buffering of the high fanout sign
extended bits of the first stage and the outputs of certain
1x1 multipliers would be accomplished. Both of the last two
improvements would be directed at optimizing the maximum
clock frequency of the multiplier.

Another possible solution to the long propagation delay
through the first stage is to partition the stage into two
stages with approximately equal delay. Although this would

59

reduce the propagation delay through the first stage, the increase in routing complexity and area required for an additional 204-bit latch may not be feasible in current MOSIS limitations.

The LSSD technique is highly recommended to be applied to any pipelined design so that the testing and detection of fabrication errors is made easier. Not only will the LSSD technique prove beneficial in the after-fabrication testing, but it also proved extremely useful in CAD simulation before fabrication to detect routing errors. The value of implementing a LSSD in most cases will far outweigh the increased complexity of the latch design and the potential frustration in searching for errors based on final latch outputs.

A 32-bit CLA adder could be developed to complement the 16-bit multiplier. This can be accomplished very rapidly and with little additional effort by using the same method described in this thesis with the following exception. Since the carry in to an adder is not necessarily zero, the equations actually input to EQNTOTT and TPLA should be Equations 2.3 through 2.8 and Equations 2.13 through 2.15. Additionally, the use of full 32-bit operands will require the expansion of all of the latches.

B. CAD HARDWARE AND SOFTWARE

The combination of EQNTOTT AND TPLA proved to be a very useful pair of CAD tools in the development of complex logic functions. Additionally, TPLA appears extremely versatile with the different technologies available and its numerous options.

CAESAR proved to be a very good design tool for the graphical layout of a VLSI design. The installation of its successor, the layout editor MAGIC, should greatly ease the routing burden of the designer.

60

The coming addition of hardware to support actual testing of chips that have been fabricated by MOSIS will greatly aid in determining the accuracy of available CAD simulation tools. Once these in-house testing capabilities are available, extensive testing should be accomplished in the two multipliers discussed here. In particular, a detailed comparison should be made between CAD simulation and actual results in the areas of functional operation, maximum speed, and static DC power consumption.

C. SILICON COMPILATION

Even though the MacPitts program available at NPS by no means provides an optimum integrated circuit design, it is an excellent vehicle from which to study the area of silicon compilers. They provide an excellent alternative to the custom, gate array, and standard cell interconnection methods that are in use today. Further research into optimizing the existing MacPitts silicon compiler for speed, power consumption, and transistor density should be undertaken.

61

APPENDIX A
STIPPLE PLOTS

On the following pages are the stipple plots of the four basic building blocks that were used in the design of the 16-bit multiplier. Following these is a stipple plot of the final layout for the 16-bit two's complement multiplier that was designed for this thesis. For the purpose of clarity and continuity, a stipple plot of the 8-bit multiplier generated by the MacPitts silicon compiler is also presented. All plots were made with the CAD program CIFPLCT.
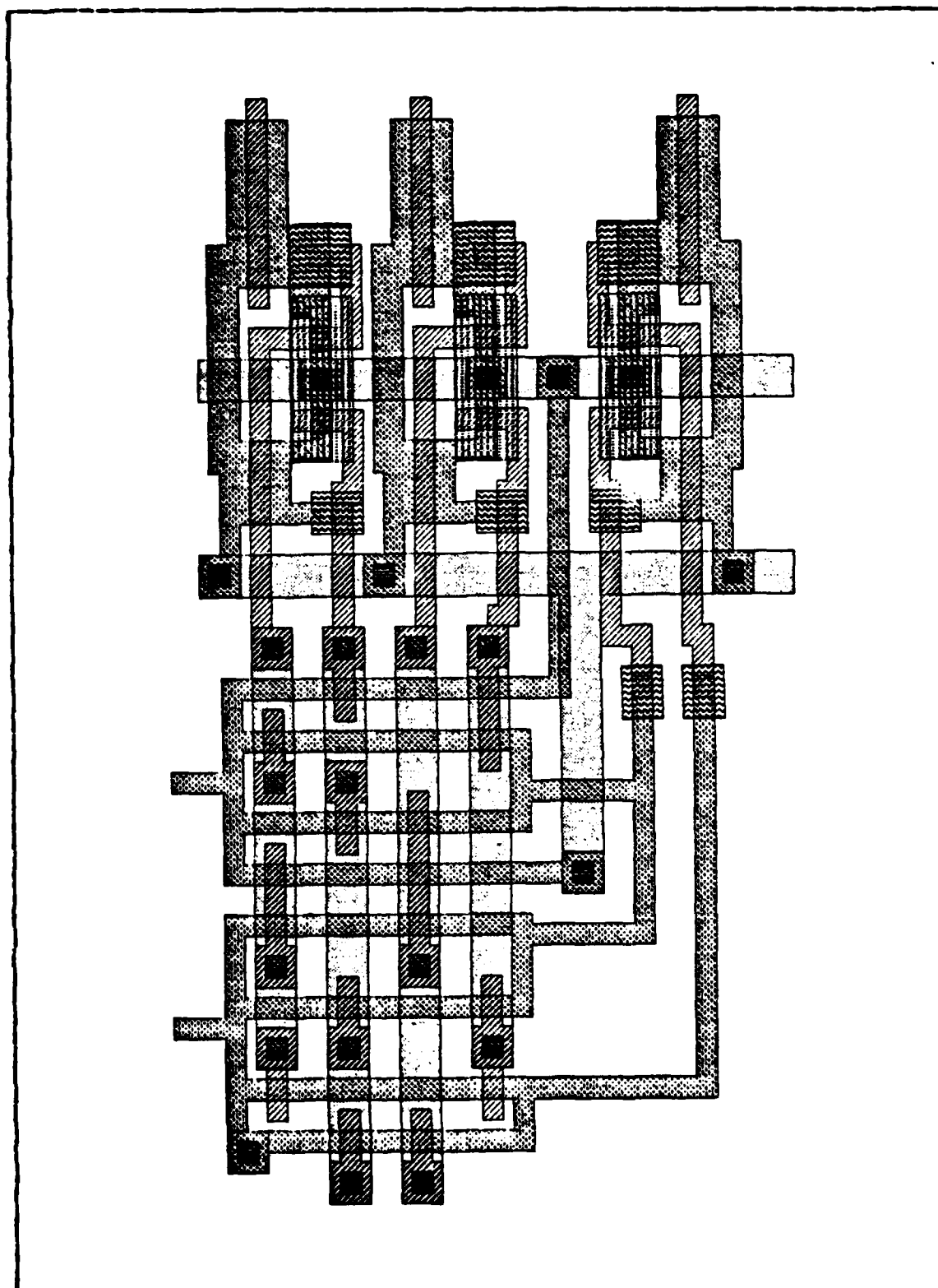
Figure A.1   Full Adder Cell.

63
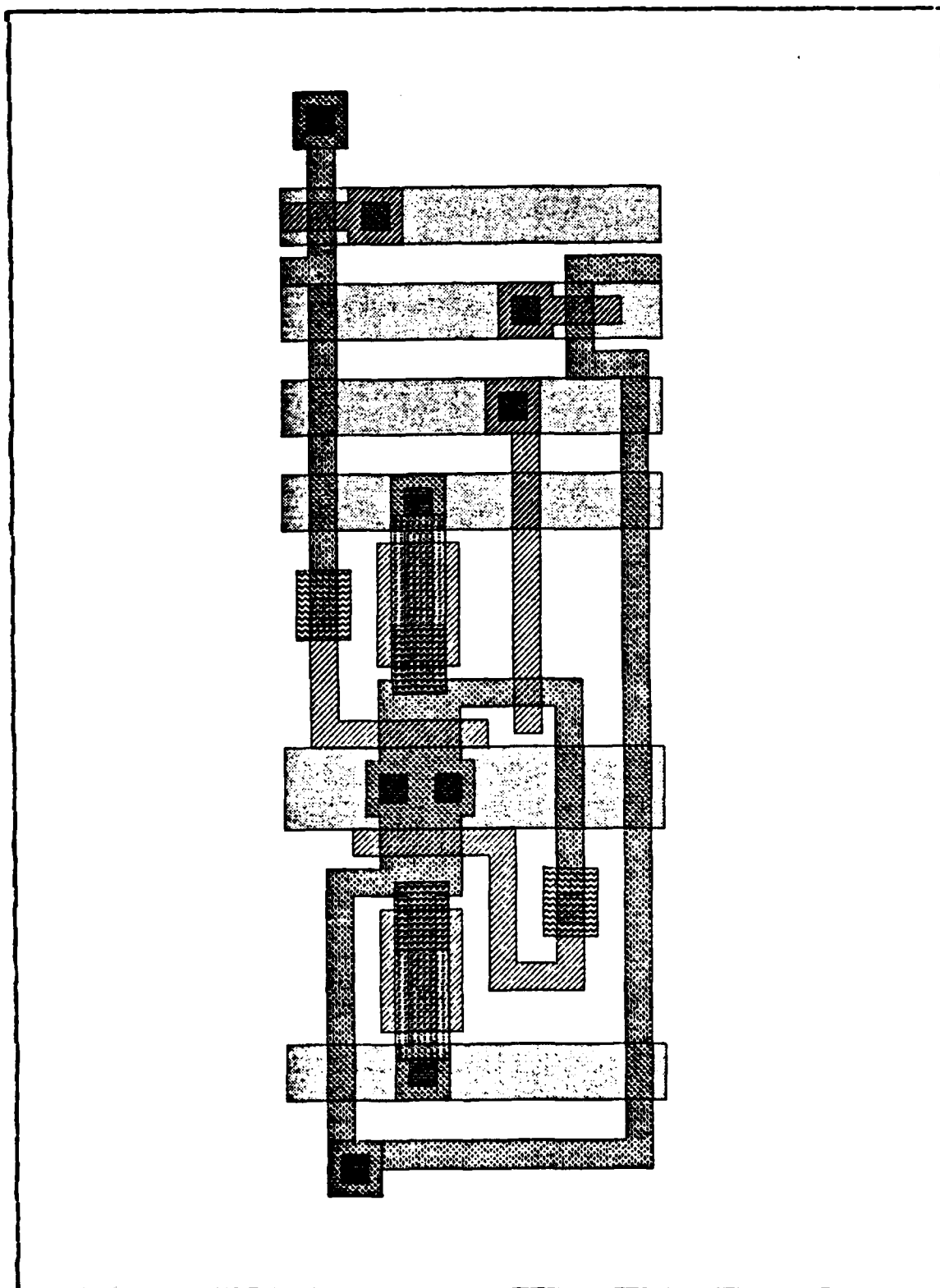
Figure A.2   1-Bit Latch Cell.

Figure A.3   CLA Unit.

65
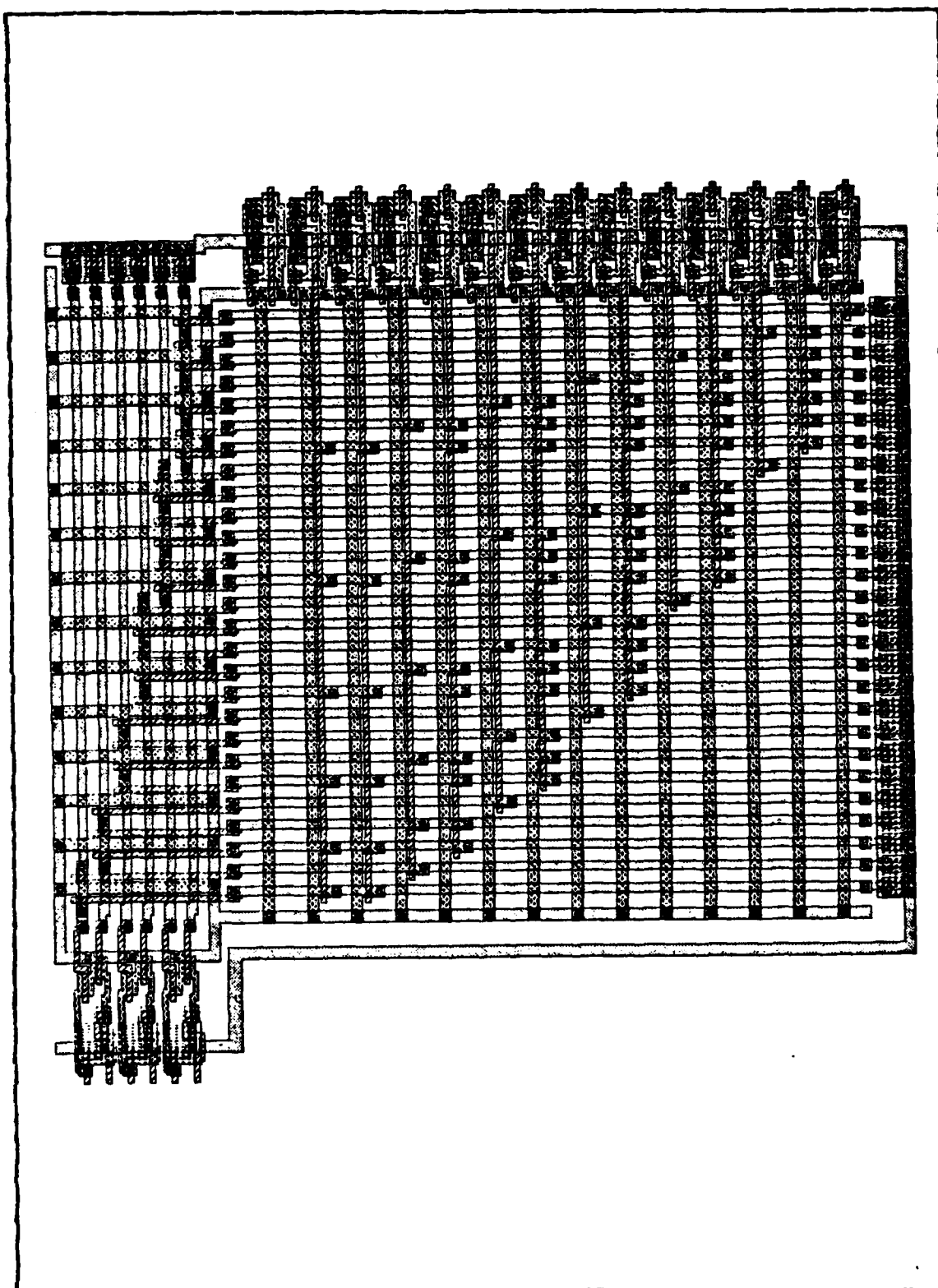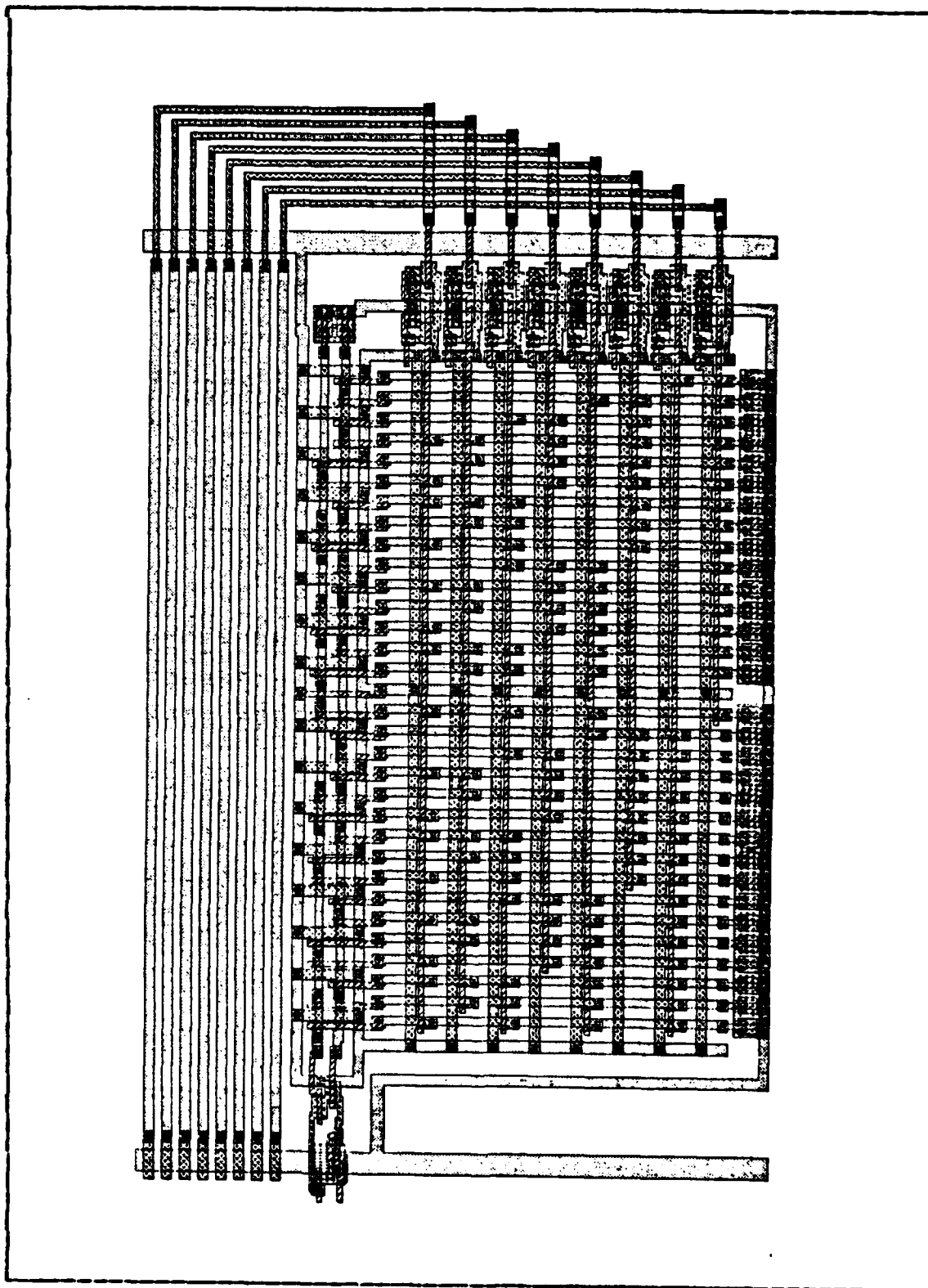
Figure A.4   Block F and G Generator.

66
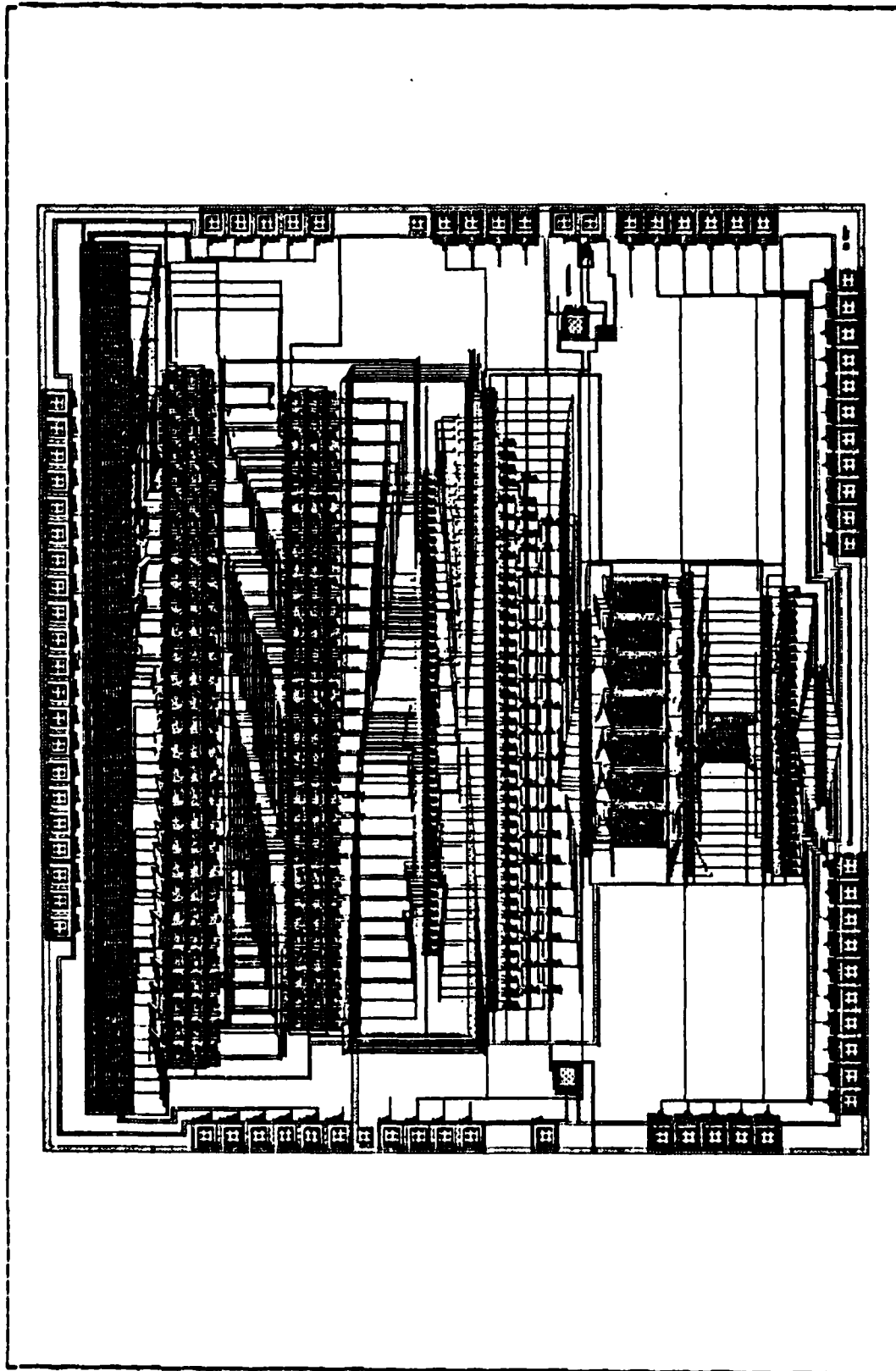
Figure A.5 Hand-crafted 16-Bit Multiplier.
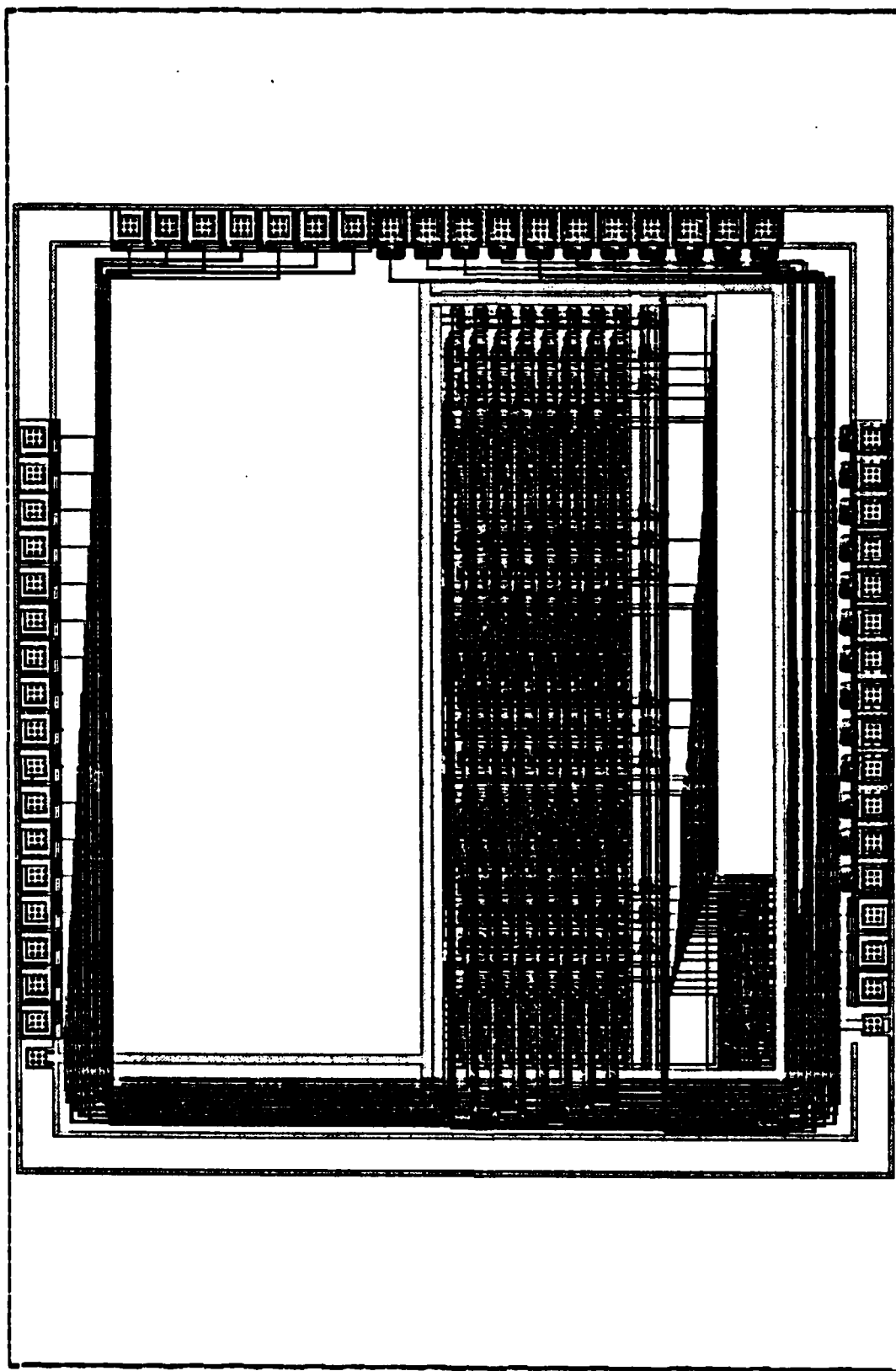
Figure A.6   MacPitts 8-Bit Multiplier.

APPENDIX B
SIMULATION RESULTS

The following pages in this appendix contain, in order, the resultant ESIM and CRYSTAL session for the 8-bit multiplier, the CRYSTAL timing analysis for the 8-bit multiplier, and the POWEST estimates for both the 16-bit and 8-bit multipliers.

ESIM results for 16-bit two's complement multiplier.

```
% esim mult32.sim
11962 transistors, 8452 nodes (3914 pulled up)

sim> @ init_esim
initialization took 33772 steps
initialization took 4682 steps
initialization took 230 steps
initialization took 0 steps
initialization took 0 steps
step took 6 events
latchout=0000   0
plow=1111111111111111   65535
phigh=1111111111111111   65535
latchin=0000   0
bin=0000000000000000   0
ain=0000000000000000   0
op=1

sim> R 5

sim> v
latchout=0000   0
plow=0000000000000000   0
phigh=0000000000000000   0
latchin=0000   0
bin=0000000000000000   0
ain=0000000000000000   0
op=1
h inputs: Vdd op
l inputs: GND phi1 phi2

sim> @ test_vector1
step took 451 events
latchout=0000   0
plow=0000000000000000   0
phigh=0000000000000000   0
latchin=0000   0
bin=0000000010001111   143
ain=0000000000011011   27
op=1
```

70

```
sim > c
latchout=0000   0
plow  0000000000000000   0
phigh=0000000000000000  0
latchin=0000    0
bin=0000000010001111   143
ain -0000000000011011   27
op=1
cycle took 3785 events

sim> @ test_vector2
step took 1927 events
latchout=0000   0
plow=0000000000000000   0
phigh=0000000000000000  0
latchin=0000    0
bin=1111111101110001   65393
ain=0000000000011011   27
op=1
sim> c
latchout=0000   0
plow=0000000000000000   0
phigh=0000000000000000  0
latchin=0000    0
bin=1111111101110001   65393
ain=0000000000011011   27
op=1
cycle took 4888 events

sim> @ test_vector3
step took 2819 events
latchout=0000   0
plow=0000000000000000   0
phigh=0000000000000000  0
latchin=0000    0
bin=0000000010001111   143
ain=1111111111100101   65509
op=1
sim> c
latchout=0000   0
plow  0000000000000000   0
phigh=0000000000000000  0
latchin=0000    0
bin=0000000010001111   143
ain=1111111111100101   65509
op=1
cycle took 5243 events
```

71

```
sim   @ test_vector4
step took 4777 events
latchout=0000    0
plow =0000000000000000   0
phigh=0000000000000000   0
latchin=0000     0
bin =1111111101110001    65393
ain=1111111111100101     65509
op=1
sim > c
latchout=0000    0
plow=0000000000000000    0
phigh=0000000000000000   0
latchin=0000     0
bin=1111111101110001     65393
ain=1111111111100101     65509
op=1
cycle took 4821 events

sim > @ test_vector5
step took 3403 events
latchout=0000    0
plow=0000000000000000    0
phigh=0000000000000000   0
latchin=0000     0
bin=0000010001100011     1123
ain=0000001101111011     891
op=1
sim > c
latchout=0000    0
plow=0000111100010101    3861
phigh=0000000000000000   0
latchin=0000     0
bin=0000010001100011     1123
ain=0000001101111011     891
op=1
cycle took 5981 events
```

72

```
sim   @ test_vector6
step. took 2121 events
latchout=-0000    0
plow = 0000111100010101    3861
phigh = 0000000000000000   0
latchin = 0000    0
bin = 0000001101111011    891
ain = 1111101110011101    64413
op = 1
sim > c
latchout = 0000    0
plow = 1111000011101011    61675
phigh = 1111111111111111   65535
latchin = 0000    0 .
bin = 0000001101111011    891
ain = 1111101110011101    64413
op = 1
cycle took 5341 events

sim > @ test_vector7
step took 1708 events
latchout = 0000    0
plow = 1111000011101011    61675
phigh = 1111111111111111   65535
latchin = 0000    0
bin = 1000000000000000    32768
ain = 1000000000000000    32768
op = 1
sim > c
latchout = 0000    0
plow = 1111000011101011    61675
phigh = 1111111111111111   65535
latchin = 0000    0
bin = 1000000000000000    32768
ain - 1000000000000000    32768
op - 1
cycle took 5084 events
```

73

```
sim > c
latchout=0000   0
plow=0000111100010101   3861
phigh=0000000000000000  0
latchin=0000    0
bin=1000000000000000    32768
ain=1000000000000000    32768
op=1
cycle took 4786 events

sim> c
latchout=0000   0
plow=0100010010010001   17553
phigh=0000000000001111  15
latchin=0000    0
bin=1000000000000000    32768
ain=1000000000000000    32768
op=1
cycle took 4170 events

sim> c
latchout=0000   0
plow=1011101101101111   47983
phigh=1111111111110000  65520
latchin=0000    0
bin=1000000000000000    32768
ain=1000000000000000    32768
op=1
cycle took 4280 events

sim> c
latchout=0000   0
plow=0000000000000000   0
phigh=0100000000000000  16384
latchin=0000    0
bin=1000000000000000    32768
ain=1000000000000000    32768
op=1
cycle took 3953 events

sim> q
```

74

CRYSTAL results for 16-bit two's complement multiplier.

```
Crystal, v.2
: build mult32.sim
[1:12.1u 0:12.4s 1786k]

: inputs a<15:0> b<15:0> op phi1 phi2
[0:00.1u 0:00.1s 1795k]
: inputs l1_in l2_in l3_in l4_in
[0:00.0u 0:00.0s 1795k]
: outputs p<31:0> l1_out l2_out l3_out l4_out
[0:00.0u 0:00.0s 1795k]

: markdynamic phi1 0 phi2 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:08.1u 0:01.1s 1795k]

*** RISETIME FOR PHI2 IN NORMAL OP ***
: set 1 op
[0:00.5u 0:00.1s 1795k]
: set 0 phi1
[0:00.7u 0:00.1s 1795k]
: delay phi2 0 -1
(12279 stages examined.)
[0:46.8u 0:04.6s 1855k]
: critical 1m
Node 14171 is driven high at 98.26ns
    ...through fet at (2772, 1751) to Vdd after
  16259 is driven low at 95.79ns
    ...through fet at (2792, 1810) to GND after
  16968 is driven high at 92.08ns
    ...through fet at (2800, 1819) to 17829
    ...through fet at (2794, 1823) to Vdd after
  1273 is driven high at 89.36ns
    ...through fet at (313, 1486) to Vdd after
  11735 is driven high at 35.90ns
    ...through fet at (303, 1506) to Vdd after
  11765 is driven high at 14.17ns
    ...through fet at (287, 1506) to Vdd after
  11745 is driven low at 10.03ns
    ...through fet at (285, 1422) to GND after
  11764 is driven high at 5.79ns
    ...through fet at (160, 1582) to Vdd after
  12847 is driven low at 0.11ns
    ...through fet at (156, 1604) to GND after
  phi2 is driven high at 0.00ns
[0:00.3u 0:00.1s 1855k]
```

75

*** FALLTIME FOR PHI2 IN NORMAL OP ***
: clear
[0:00.9u 0:00.3s 1855k]
: set 1 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.4u 0:00.6s 1855k]
: set 0 phi1
[0:00.8u 0:00.1s 1855k]
: delay phi2 -1 0
(16400 stages examined.)
[0:58.8u 0:02.6s 1879k]
: critical 1m
Node 11983 is driven low at 64.28ns
        ...through fet at (2836, 1550) to GND after
    12776 is driven high at 64.98ns
        ...through fet at (2842, 1602) to 13219
        ...through fet at (2852, 1602) to 13220
        ...through fet at (2863, 1645) to Vdd after
    12892 is driven high at 54.01ns
        ...through fet at (2840, 1645) to Vdd after
    13081 is driven low at 53.08ns
        ...through fet at (2836, 1656) to GND after
    14010 is driven high at 55.67ns
        ....through fet at (2756, 1696) to 14572
        ...through fet at (2772, 1696) to 14437
        ...through fet at (2782, 1751) to Vdd after
    14171 is driven low at 35.54ns
        ...through fet at (2767, 1756) to GND after
    16259 is driven high at 33.63ns
        ...through fet at (2794, 1800) to Vdd after
    16968 is driven low at 22.80ns
        ...through fet at (2800, 1819) to 17829
        ...through fet at (2792, 1816) to GND after
    1273 is driven high at 21.54ns
        ...through fet at (313, 1486) to Vdd after
    11735 is driven high at 13.39ns
        ...through fet at (293, 1506) to Vdd after
    11765 is driven low at 10.69ns
        ...through fet at (285, 1483) to GND after
    11745 is driven high at 7.19ns
        ...through fet at (287, 1410) to Vdd after
    11764 is driven low at 2.51ns
        ...through fet at (156, 1581) to GND after
    12847 is driven high at 0.56ns
        ...through fet at (163, 1604) to Vdd after
    phi2 is driven low at 0.00ns
[0:00.3u 0:00.1s 1879k]

```
*** PHI1 RISETIME IN NORMAL OP ***
: clear
[0:00.9u 0:00.3s 1879k]
: set 1 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.5u 0:00.5s 1879k]
: set 0 phi2
[0:00.2u 0:00.0s 1879k]
: delay phi1 0 -1
(5926 stages examined.)
[0:12.1u 0:00.6s 1879k]
: critical 1m
Node 17518 is driven high at 108.62ns
    ...through fet at (2256, 1845) to 18827 after
  normdrout is driven high at 101.60ns
    ...through fet at (4013, 1343) to Vdd after
  10876 is driven high at 48.60ns
    ...through fet at (4141, 1351) to Vdd after
  11000 is driven high at 26.81ns
    ...through fet at (4163, 1351) to Vdd after
  11302 is driven low at 22.55ns
    ...through fet at (4166, 1423) to GND after
  11063 is driven high at 17.47ns
    ...through fet at (4408, 1354) to Vdd after
  11064 is driven low at 6.61ns
    ...through fet at (4433, 1362) to 11369
    ...through fet at (4433, 1366) to GND after
  10622 is driven high at 5.72ns
    ...through fet at (4483, 1305) to Vdd after
  10603 is driven low at 0.11ns
    ...through fet at (4498, 1281) to GND after
  phi1 is driven high at 0.00ns
[0:00.1u 0:00.1s 1879k]


*** PHI1 FALLTIME FOR NORMAL OP ***
: clear
[0:00.8u 0:00.3s 1879k]
: set 1 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.2u 0:00.1s 1879k]
: set 0 phi2
[0:00.2u 0:00.0s 1879k]
: delay phi1 -1 0
(4092 stages examined.)
[0:10.4u 0:00.6s 1896k]
```

: critical 1m
Node 4675 is driven low at 89.11ns
　　　...through fet at (2091, 781) to GND after
　　4486 is driven high at 83.87ns
　　　...through fet at (2736, 842) to Vdd after
　　normdrout is driven low at 39.96ns
　　　...through fet at (4021, 1351) to GND after
　　11059 is driven high at 32.15ns
　　　...through fet at (4141, 1446) to Vdd after
　　11302 is driven high at 10.54ns
　　　...through fet at (4163, 1446) to Vdd after
　　11063 is driven low at 5.91ns
　　　...through fet at (4407, 1362) to GND after
　　11064 is driven high at 3.13ns
　　　...through fet at (4434, 1354) to Vdd after
　　10622 is driven low at 2.49ns
　　　...through fet at (4498, 1304) to GND after
　　10603 is driven high at 0.56ns
　　　...through fet at (4489, 1282) to Vdd after
　　phi1 is driven low at 0.00ns
[0:00.2u 0:00.1s 1896k]

*** PHI1 RISETIME FOR SHIFT OP ***
: clear
[0:00.9u 0:00.3s 1896k]
: set 0 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.6u 0:00.5s 1896k]
: set 0 phi2
[0:00.2u 0:00.0s 1896k]
: delay phi1 0 -1
(11989 stages examined.)
[0:42.1u 0:01.7s 1918k]
: critical 1m
Node 4354 is driven high at 343.02ns
　　　...through fet at (2743, 502) to 3227
　　　...through fet at (2734, 463) to Vdd after
　　shdrout is driven high at 45.78ns
　　　...through fet at (4007, 1223) to Vdd after
　　10522 is driven low at 29.07ns
　　　...through fet at (4053, 1228) to GND after
　　10336 is driven high at 27.68ns
　　　...through fet at (4067, 1216) to Vdd after
　　10523 is driven low at 24.23ns
　　　...through fet at (4070, 1266) to GND after

78

10589 is driven high at 19.49ns
  ...through fet at (4407, 1334) to Vdd after
10633 is driven low at 6.61ns
  ...through fet at (4433, 1327) to 10631
  ...through fet at (4433, 1324) to GND after
10622 is driven high at 5.72ns
  ...through fet at (4483, 1305) to Vdd after
10603 is driven low at 0.11ns
  ...through fet at (4498, 1281) to GND after
phi1 is driven high at 0.00ns
[0:00.1u 0:00.1s 1918k]

*** PHI1 FALLTIME FOR A SHIFT OP ***
: clear
[0:00.8u 0:00.3s 1918k]
: set 0 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.4u 0:00.4s 1918k]
: set 0 phi2
[0:00.2u 0:00.0s 1918k]
: delay phi1 -1 0
(20633 stages examined.)
[1:22.2u 0:08.4s 1961k]
: critical 1m
Node 11983 is driven low at 72.04ns
    ...through fet at (2836, 1550) to GND after
  12776 is driven high at 70.74ns
    ...through fet at (2842, 1602) to 13219
    ...through fet at (2852, 1602) to 13220
    ...through fet at (2863, 1645) to Vdd after
  12892 is driven high at 61.78ns
    ...through fet at (2840, 1645) to Vdd after
  13081 is driven low at 60.84ns
    ...through fet at (2836, 1656) to GND after
  14010 is driven high at 63.43ns
    ...through fet at (2756, 1696) to 14572
    ...through fet at (2772, 1696) to 14437
    ...through fet at (2782, 1751) to Vdd after
  14171 is driven low at 43.30ns
    ...through fet at (2767, 1756) to GND after
  16259 is driven high at 41.39ns
    ...through fet at (2794, 1800) to Vdd after
  shdrout is driven low at 30.70ns
    ...through fet at (4032, 1225) to GND after

79

10522 is driven high at 19.22ns
　　...through fet at (4045, 1289) to Vdd after
10523 is driven high at 10.01ns
　　...through fet at (4067, 1289) to Vdd after
10589 is driven low at 6.29ns
　　...through fet at (4406, 1327) to GND after
10633 is driven high at 3.13ns
　　...through fet at (4434, 1334) to Vdd after
10622 is driven low at 2.49ns
　　...through fet at (4498, 1304) to GND after
10603 is driven high at 0.56ns
　　...through fet at (4489, 1282) to Vdd after
phi1 is driven low at 0.00ns
[0:00.2u 0:00.2s 1961k]

*** INPUT PAD TO LATCH 1 DELAY ***
: clear
[0:00.9u 0:00.7s 1961k]
: set 1 op
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:06.3u 0:00.3s 1961k]
: set 0 phi1 phi2
[0:00.9u 0:00.1s 1961k]
: delay a<15:0> 0 0
(43921 stages examined.)
[2:16.6u 0:09.2s 1961k]
: critical 1m
Node 19554 is driven high at 558.82ns
　　　...through fet at (1008, 2140) to Vdd after
　19655 is driven low at 554.42ns
　　　...through fet at (980, 2145) to GND after
　21705 is driven high at 531.79ns
　　　...through fet at (667, 2760) to 27839
　　　...through fet at (677, 2760) to 27714
　　　...through fet at (693, 2798) to Vdd after
　27436 is driven low at 485.22ns
　　　...through fet at (698, 2808) to GND after
　22366 is driven high at 473.40ns
　　　...through fet at (1823, 3125) to Vdd after
　30352 is driven low at 337.44ns
　　　...through fet at (183⌐ 3142) to GND after
　30351 is driven high at 332.90ns
　　　...through fet at (1807, 3257) to 33567
　　　...through fet at (1817, 3257) to 33568
　　　...through fet at (1840, 3306) to Vdd after

80

```
        33186 is driven low at 299.22ns
          ...through fet at (1818, 3293) to GND after
        33391 is driven high at 298.15ns
          ...through fet at (1822, 3306) to Vdd after
        30591 is driven low at 295.49ns
          ...through fet at (1955, 3577) to 38872
          ...through fet at (1955, 3580) to GND after
        38615 is driven high at 241.93ns
          ...through fet at (1997, 3813) to Vdd after
        40527 is driven low at 3.37ns
          ...through fet at (2011, 3839) to GND after
        40457 is driven high at 2.61ns
          ...through fet at (2030, 3824) to Vdd after
        40625 is driven low at 0.11ns
          ...through fet at (2052, 3839) to GND after
        a2 is driven high at 0.00ns
[0:00.2u 0:00.2s 1961k]


: q
[8:58.2u 0:49.0s 1961k] Crystal done.
```

```
 build stage1.sim
[0 12.4u 0:01.3s 247k]
: inputs in<27:1>
[0:00.0u 0:00.1s 256k]
: outputs a<24:1>

*** FIRST STAGE DELAY ***
: delay in<27:1> 0 0
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
(11559 stages examined.)
[0:22.7u 0:00.9s 411k]
: critical
Node 2195 is driven high at 4838.89ns
     ...through fet at (565, 934) to Vdd after
   2118 is driven low at 4831.44ns
     ...through fet at (506, 926) to 2127
     ...through fet at (506, 921) to GND after
   2095 is driven high at 4825.41ns
     ...through fet at (485, 928) to Vdd after
   1867 is driven low at 4813.82ns
     ...through fet at (423, 922) to 2086
     ...through fet at (423, 917) to GND after
   1805 is driven high at 4783.75ns
     ...through fet at (669, 910) to a2
     ...through fet at (683, 910) to 1944
     ...through fet at (620, 934) to Vdd after
   2119 is driven low at 4330.98ns
     ...through fet at (585, 924) to 2103
     ...through fet at (585, 919) to GND after
   2048 is driven high at 4326.95ns
     ...through fet at (537, 930) to Vdd after
   1933 is driven low at 4314.44ns
     ...through fet at (645, 1000) to 2790
     ...through fet at (645, 1005) to GND after
   2730 is driven high at 4306.41ns
     ...through fet at (537, 1010) to Vdd after
   2798 is driven low at 4293.82ns
     ...through fet at (506, 1006) to 2807
     ...through fet at (506, 1001) to GND after
   2775 is driven high at 4287.69ns
     ...through fet at (485, 1008) to Vdd after
   2551 is driven low at 4275.76ns
     ...through fet at (423, 1002) to 2766
     ...through fet at (423, 997) to GND after
```

2525 is driven high at 4243.64ns
    ...through fet at (669, 990) to a3
    ...through fet at (683, 990) to 2637
    ...through fet at (620, 1014) to Vdd after
2799 is driven low at 3741.79ns
    ...through fet at (585, 1004) to 2783
    ...through fet at (585, 999) to GND after
2624 is driven high at 3735.64ns
    ...through fet at (652, 1074) to Vdd after
3236 is driven low at 3712.11ns
    ...through fet at (423, 1082) to 3449
    ...through fet at (423, 1077) to GND after
3210 is driven high at 3680.28ns
    ...through fet at (669, 1070) to a4
    ...through fet at (683, 1070) to 3318
    ...through fet at (620, 1094) to Vdd after
3482 is driven low at 3186.59ns
    ...through fet at (585, 1084) to 3466
    ...through fet at (585, 1079) to GND after
3411 is driven high at 3182.56ns
    ...through fet at (537, 1090) to Vdd after
3307 is driven low at 3170.04ns
    ...through fet at (645, 1160) to 4149
    ...through fet at (645, 1165) to GND after
4087 is driven high at 3162.01ns
    ...through fet at (537, 1170) to Vdd after
4157 is driven low at 3149.43ns
    ...through fet at (506, 1166) to 4166
    ...through fet at (506, 1161) to GND after
4133 is driven high at 3143.25ns
    ...through fet at (485, 1168) to Vdd after
3907 is driven low at 3131.21ns
    ...through fet at (423, 1162) to 4124
    ...through fet at (423, 1157) to GND after
3881 is driven high at 3098.30ns
    ...through fet at (669, 1150) to a5
    ...through fet at (683, 1150) to 3990
    ...through fet at (620, 1174) to Vdd after
4158 is driven low at 2577.22ns
    ...through fet at (585, 1164) to 4141
    ...through fet at (585, 1159) to GND after
3978 is driven high at 2571.91ns
    ...through fet at (652, 1234) to Vdd after
4770 is driven low at 2555.05ns
    ...through fet at (530, 1244) to 4825
    ...through fet at (530, 1239) to GND after
4841 is driven high at 2547.85ns
    ...through fet at (513, 1252) to Vdd after

4818 is driven low at 2532.70ns
  ...through fet at (478, 1242) to 4810
  ...through fet at (478, 1237) to GND after
4568 is driven high at 2501.33ns
  ...through fet at (669, 1230) to a6
  ...through fet at (683, 1230) to 4677
  ...through fet at (620, 1254) to Vdd after
4842 is driven low at 1985.61ns
  ...through fet at (585, 1244) to 4826
  ...through fet at (585, 1239) to GND after
4666 is driven high at 1980.29ns
  ...through fet at (652, 1314) to Vdd after
5456 is driven low at 1963.43ns
  ...through fet at (530, 1324) to 5508
  ...through fet at (530, 1319) to GND after
5526 is driven high at 1956.23ns
  ...through fet at (513, 1332) to Vdd after
5501 is driven low at 1941.04ns
  ...through fet at (478, 1322) to 5493
  ...through fet at (478, 1317) to GND after
5248 is driven high at 1909.46ns
  ...through fet at (669, 1310) to a7
  ...through fet at (683, 1310) to 5363
  ...through fet at (620, 1334) to Vdd after
5527 is driven low at 1388.69ns
  ...through fet at (585, 1324) to 5509
  ...through fet at (585, 1319) to GND after
5346 is driven high at 1383.38ns
  ...through fet at (652, 1394) to Vdd after
6129 is driven low at 1366.51ns
  ...through fet at (530, 1404) to 6181
  ...through fet at (530, 1399) to GND after
6197 is driven high at 1359.33ns
  ...through fet at (513, 1412) to Vdd after
6174 is driven low at 1344.20ns
  ...through fet at (478, 1402) to 6166
  ...through fet at (478, 1397) to GND after
5928 is driven high at 1312.98ns
  ...through fet at (669, 1390) to a8
  ...through fet at (683, 1390) to 6036
  ...through fet at (620, 1414) to Vdd after
6198 is driven low at 800.61ns
  ...through fet at (585, 1404) to 6182
  ...through fet at (585, 1399) to GND after
6025 is driven high at 794.45ns
  ...through fet at (652, 1474) to Vdd after
6637 is driven low at 770.92ns
  ...through fet at (423, 1482) to 6842
  ...through fet at (423, 1477) to GND after

84

```
6611 is driven high at 739.09ns
  ...through fet at (669, 1470) to 6644
  ...through fet at (683, 1470) to 6720
  ...through fet at (620, 1494) to Vdd after
755 is driven high at 219.87ns
  ...through fet at (634, 410) to Vdd after
1080 is driven low at 134.69ns
  ...through fet at (2443, 2876) to GND after
7571 is driven high at 10.74ns
  ...through fet at (2487, 2858) to Vdd after
in16 is driven low at 0.00ns
[0:00.7u 0:00.4s 411k]


: q
```

CRYSTAL results for the clock inputs to
the registers of the Macpitts chip.

```
Crystal, v.2
: build timing.sim
[0:13.9u 0:01.6s 258k]

: inputs phia phib phic
[0:00.0u 0:00.0s 267k]

*** PHASE 1 OF 5 ***
: set 1 phia phic
[0:00.1u 0:00.0s 267k]
: delay phib 0 -1
(604 stages examined.)
[0:00.9u 0:00.1s 271k]
: critical
Node 6392 is driven low at 87.36ns
      ...through fet at (2322, 1476) to 6678
      ...through fet at (2314, 1472) to GND after
   6391 is driven high at 81.45ns
      ...through fet at (2290, 1485) to 6679
      ...through fet at (2333, 1483) to Vdd after
   588 is driven high at 65.23ns
      ...through fet at (2316, 841) to Vdd after
   490 is driven low at 62.98ns
      ...through fet at (2314, 834) to GND after
   28 is driven high at 50.57ns
      ...through fet at (791, 149) to Vdd after
   21 is driven low at 0.80ns
      ...through fet at (817, 134) to GND after
   phib is driven high at 0.00ns
[0:00.1u 0:00.1s 271k]

*** PHASE 2 OF 5 ***
: clear
[0:00.1u 0:00.0s 271k]
: set 1 phia
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:00.6u 0:00.0s 271k]
: delay phib -1 0
(28 stages examined.)
[0:00.1u 0:00.0s 271k]
: delay phic -1 0
(28 stages examined.)
[0:00.1u 0:00.0s 271k]
```

critical
Node 590 is driven low at 119.19ns
    ...through fet at (2344, 833) to GND after
  491 is driven high at 113.28ns
    ...through fet at (2338, 813) to Vdd after
  25 is driven low at 84.73ns
    ...through fet at (651, 134) to GND after
  19 is driven high at 10.74ns
    ...through fet at (695, 148) to Vdd after
  phic is driven low at 0.00ns
[0:00.1u 0:00.0s 271k]

***PHASE 3 OF 5 ***
: clear
[0:00.1u 0:00.0s 271k]
: set 0 phib phic
[0:00.1u 0:00.0s 271k]
: delay phia -1 0
(40 stages examined.)
[0:00.1u 0:00.0s 272k]
: critical
Node 574 is driven high at 61.22ns
    ...through fet at (2087, 841) to Vdd after
  483 is driven low at 59.11ns
    ...through fet at (2085, 834) to GND after
  353 is driven high at 49.97ns
    ...through fet at (2088, 802) to Vdd after
  31 is driven low at 30.89ns
    ...through fet at (907, 134) to GND after
  23 is driven high at 10.74ns
    ...through fet at (951, 148) to Vdd after
  phia is driven low at 0.00ns
[0:00.1u 0:00.1s 272k]

*** PHASE 4 OF 5 ***
: clear
[0:00.1u 0:00.0s 272k]
: set 0 phib phic
[0:00.1u 0:00.0s 272k]
: delay phia 0 -1
(40 stages examined.)
[0:00.1u 0:00.0s 274k]
: critical
Node 574 is driven low at 54.31ns
    ...through fet at (2095, 833) to GND after

483 is driven high at 49.17ns
    ...through fet at (2089, 813) to Vdd after
353 is driven low at 27.72ns
    ...through fet at (2082, 792) to GND after
31 is driven high at 15.16ns
    ...through fet at (919, 149) to Vdd after
23 is driven low at 0.80ns
    ...through fet at (945, 134) to GND after
phia is driven high at 0.00ns
[0:00.1u 0:00.0s 274k]

*** PHASE 5 OF 5 ***
: clear
[0:00.1u 0:00.0s 274k]
: set 1 phia
Marking transistor flow...
Setting Vdd to 1...
Setting GND to 0...
[0:00.6u 0:00.1s 274k]
: set 0 phib
[0:00.1u 0:00.0s 274k]
: delay phic 0 -1
(412 stages examined.)
[0:00.5u 0:00.0s 281k]
: critical
Node 6674 is driven low at 91.61ns
    ...through fet at (2136, 1472) to GND after
  6384 is driven high at 85.13ns
    ...through fet at (2116, 1476) to 6673
    ...through fet at (2099, 1483) to Vdd after
  578 is driven high at 70.69ns
    ...through fet at (2130, 841) to Vdd after
  485 is driven low at 68.51ns
    ...through fet at (2128, 834) to GND after
  25 is driven high at 55.79ns
    ...through fet at (663, 149) to Vdd after
  19 is driven low at 0.80ns
    ...through fet at (689, 134) to GND after
  phic is driven high at 0.00ns
[0:00.1u 0:00.0s 281k]
: q

POWEST Results for the 16-bit Multiplier

% powest -p < mult32.sim

gamma=0.4V**.5, tox=9e-08m, u0=0.08m**2/V-s
vdd=5V, vtd=-3.5V, vte=0.8V, vsb=2V

#devs	Pdc_avg (W)	Pdc_max (W)	type
0	0.000000	0.000000	enhancement pullups
3720	1.790881	2.793533	depletion pullups
194	0.191948	0.383896	special depletion pullups
3914	1.982829	3.177428	TOTAL

POWEST Results for the 8-bit Multiplier.

% powest -p < multip8c4.sim

gamma=0.4V**.5, tox=9e-08m, u0=0.08m**2/V-s
vdd=5V, vtd=-3.5V, vte=0.8V, vsb=2V

#devs	Pdc_avg (W)	Pdc_max (W)	type
0	0.000000	0.000000	enhancement pullup
690	0.140672	0.244640	depletion pullups
111	0.211404	0.422809	special depletion pullups
801	0.352076	0.667449	TOTAL

APPENDIX C
TEST VECTORS

This appendix contains the inputs, intermediate latch values, and the final product output for each of the test vector pairs described in Chapter 3. Each binary value is represented as its hexadecimal equivalent. The inputs and outputs are represented with their most significant hexadecimal digit in the leftmost position. The intermediate latch contents are represented in hexadecimal with the Nth bit shifted out of the latch and placed to the left of the previous bit serially shifted out. The latch at the end of stage X is identified as latchX where X goes from 1 to 4.

TEST_VECTOR_1

```
INPUTS:    001B      008F
OUTPUT:    00000F15
LATCH1:    00000000000000000000000000011072E7
LATCH2:    00000000002A6E7
LATCH3:    0000000000000153DC7
LATCH4:    000000000CA9DC7
```

TEST_VECTOR_2

```
INPUTS:    FF71      0C1B
OUTPUT:    FFFFF0EB
LATCH1:    65965965965965965965965969768C0D5A7295
LATCH2:    155555554AEE695
LATCH3:    2A9AA6A9AA15D70D15
LATCH4:    AA552595457B8D15
```

TEST_VECTOR_3

```
INPUTS:   008F        FFE5
OUTPUT:   FFFFF0EB
LATCH1:   4104104104104010406016C90B6062250A53
LATCH2:   15555555530653
LATCH3:   2A9AA6A9AA2A580C93
LATCH4:   AA552A954ASC0C93
```

TEST_VECTOR_4

```
INPUTS:   FFE5        FF71
OUTPUT:   00000F15
LATCH1:   4F3CF3CF3CF7CA38E768B6C85B49E01EB429
LATCH2:   0AAB34D5562A829
LATCH3:   15559699AAAC354049
LATCH4:   55ACE9B55E0AA049
```

TEST_VECTOR_5

```
INPUTS:   0463        037B
OUTPUT:   000F4491
LATCH1:   000000000000000000020800884A01F82641F
LATCH2:   00000014A1C641F
LATCH3:   0000001A50383083F
LATCH4:   00000014A0E1883F
```

TEST_VECTOR_6

```
INPUTS:   037B        FE9D
OUTPUT:   FFF0BB6F
LATCH1:   4104104104C0450653245514549A2B30F169B
LATCH2:   15552C756994A9B
LATCH3:   2A9A918FAB130A451B
LATCH4:   AA5491F564C5251B
```

91

TEST_VECTOR 7

INPUTS: 8000   8000
OUTPUT: 40000000
LATCH1:   0004 104 104 104 104 104 104 104 12000000000
LATCE2:   015555555 8C0000
LATCH3:   029 AA6A9AAE0000000
LATCH4:   0AD56AB55CC00000

LIST OF REFERENCES

1. Mead, C. and Conway, L., _Introduction to VLSI Systems_, Addison-Wesley, 1980.

2. Carlson, D.J., _Application of a Silicon Compiler to VLSI Design of Digital Pipelined Multipliers_, MSEE Thesis, Naval Postgraduate School, Monterey, California, June 1984.

3. Stone, H.S., _Introduction to Computer Architecture_, 2d ed., pp. 29-90, Science Research Associates, 1980.

4. Waser, S. and Flynn, M.J., _Introduction to Arithmetic for Digital System Designers_, CBS College Publishing, 1982.

5. Hwang, K. _Computer Arithmetic Principles, Architecture, and Design_, Wiley, 1979.

6. Kogge, P.M., _The Architecture of Pipelined Computers_, Hemisphere Publishing, 1981.

7. Reid, William R., _Design of a Sixteen Bit Pipelined Adder Using CMOS Bulk P-Well Technology_, MSEE Thesis, Naval Postgraduate School, Monterey, California, December 1984.

8. Ousterhout, J., _Editing VLSI Circuits with Caesar_, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, pp. 1-22, March 22, 1983.

9. Computer Science Division (EECS), University of California, Berkeley, Report No. UCB/CSD/83/115, 1983 _VLSI Tools_, edited by R.N. Mayo, J.K. Ousterhout, and W.J. Scott, March, 1983.

10. Ousterhout, J., _Using Crystal for Timing Analysis_, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, pp. 1-23, February 28, 1985.

11. Froede, A., _Silicon Compiler Design of Combinational and Pipeline Adder Integrated Circuits_, MSEE Thesis, Naval Postgraduate School, Monterey, California, June 1985.

12. Newkirk, J., _VLSI System Design_, paper presented at Eighteenth Annual Asilomar Conference on Circuits, Systems, and Computers, Monterey, California, 6 November 1984.

93

INITIAL DISTRIBUTION LIST

		No.	Ccpies
1.	Superintendent Attn: Library, Code 0142 Naval Postgraduate School Monterey, Califcrnia 93943-5100		2
2.	Dr. Herschel H. Ioomis Code 62LM Naval Postgraduate School Monterey, Califcrnia 93943-5100		5
3.	Dr. Donald Z. Kirk Code 62KI Naval Postgraduate School Monterey, Califcrnia 93943-5100		2
4.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145		2
5.	CPT Richard J. Simchik Jr. 594 Genesee Street Oneida, New York 13421		1

END

FILMED

11-85

DTIC